DTIC FILE COPY

4

[illegible]-11, Vol VIII (of eight) Part B
[illegible] Technical Report

AD-A198 597

NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT 1986 Parallel, Structural, and Optimal Techniques in Vision

Syracuse University

Christopher M. Brown, et. al.

This effort was funded partially by the Laboratory Director's fund.

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss AFB, NY 13441-5700

88

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-88-11, Volume VIII (of eight), Part B has been reviewed and is approved for publication.

APPROVED:

MICHAEL D. RICHARD, Captain, USAF
Project Engineer

APPROVED:

GARRY W. BARRINGER
Technical Director
Directorate of Intelligence & Reconnaissance

FOR THE COMMANDER:

JAMES W. HYDE III
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (IRRE) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-88-11, Vol VIII (of eight), Part B
6a. NAME OF PERFORMING ORGANIZATION Northeast Artificial Intelligence Consortium (NAIC) 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COES)
6c. ADDRESS (City, State, and ZIP Code) 409 Link Hall Syracuse University Syracuse NY 13244-1240	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center 8b. OFFICE SYMBOL (If applicable) COES	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-85-C-0008
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 62702F (over) PROJECT NO: 5581 TASK NO: 27 WORK UNIT ACCESSION NO: 13

11. TITLE (Include Security Classification)
NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT 1986 Parallel, Structural, and Optimal Techniques in Vision

12. PERSONAL AUTHOR(S)
Christopher M. Brown, et.al.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Interim	FROM Jan 86 TO Dec 86	June 1988	238

16. SUPPLEMENTARY NOTATION
This effort was performed as a subcontract by the University of Rochester to Syracuse University, Office of Sponsored Programs. (See reverse)

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
12	05	
25	05	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Artificial Intelligence, Temporal Reasoning, Planning, Plan Recognition, Computer Vision, Parallel Computation, Multiprocessing, Edge Detection.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Northeast Artificial Intelligence Consortium (NAIC) was created by the Air Force Systems Command, Rome Air Development Center, and the Office of Scientific Research. Its purpose is to conduct pertinent research in artificial intelligence and to perform activities ancillary to this research. These volumes describe progress that has been made in the second year of the existence of the NAIC on the technical research tasks undertaken at the member universities. The topics covered in general are: versatile expert system for equipment maintenance, distributed AI for communications system control, automatic photo interpretation, time-oriented problem solving, speech understanding systems, knowledge base maintenance, hardware architectures for very large systems, knowledge-based reasoning and planning, and a knowledge acquisition, assistance, and explanation system. The specific topic for PART A of this volume is a model theory and axiomatization of a logic for reasoning about planning in domains of concurrent actions. PART B addresses various aspects of parallel, structural, and optimal techniques in computer vision.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Michael D. Richard	22b. TELEPHONE (Include Area Code) (315) 330-7788 22c. OFFICE SYMBOL RADC (IRRE)

DD Form 1473, JUN 86 Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

UNCLASSIFIED

Block 10 (Cont'd)

Program Element Number	Project Number	Task Number	Work Unit Number
62702F	4594	18	E2
61101F	LDFP	15	C4
61102F	2304	J5	01
33126F	2155	02	10

Block 16 (Cont'd)

This effort was funded partially by the Laboratory Directors' Fund.

UNCLASSIFIED

Part B: Parallel, Structural and Optimal Techniques in Vision

Christopher M. Brown
Computer Science Department
University of Rochester
Rochester, NY 14627

Table of Contents

B.1 Computer Vision and Structure

Paul Cooper's goal is to do object recognition, using structural (relational) information about the object rather than global properties such as shape [Cooper 1987]. The other characteristic that sets this work apart is the large database of models from which identification is to take place. This work is applicable to any object-recognition situation in which the relations of the parts form an appreciable part of the semantics of the configuration. For instance, it could be used to classify arrangements of units deployed in a tactical situation into such classes as "convoy," "patrol," "defensive line," etc.

The work has taken three main paths simultaneously:

1) Development of a prototype end-to-end system, experimentation with it, and documentation of results.

2) Work on stereo from structure.

3) Work on uncertainty in recognition from structure.

Each is expanded upon briefly below.

B.1.1 Prototype end-to-end System

Work on this system was begun in the spring of 86, with Susan Hollbach, Nigel Goddard, and Jerry Feldman [Hollbach 1986]. In the fall and winter, Cooper's primary focus was upon making some major improvements to the algorithm, establishing a broader rationale for the approach taken, performing some detailed analysis of the expected performance of the algorithm, and performing a long series of much more comprehensive experiments. Some modules of the algorithm were implemented in parallel upon the butterfly parallel processor by Steve Whitehead.

A paper documenting the results was submitted to ICCV and AAAI-87. The paper as well as a videotape and operational demo of the system were presented at the 1987 DARPA Image Understanding Workshop [Cooper and Hollbach 1987]. Overall, we now have a good idea of the capabilities and limitations of the system.

B.1.2 Stereo from Structure

A potential requirement for 3D input into the recognition process prompted work on porting a stereo algorithm Cooper developed in other work [Cooper et al. 1985]. After extensive recoding and porting, the system is now operational and easily usable on our new Sun3 with floating point accelerator. In [Cooper 1987], Cooper explored the relationship between the domain and the underlying principles that made the stereo work successful. This paper was prepared in December/January 1986.

B.1.3 Uncertainty in Recognition from Structure

The necessity of working with imperfect and incomplete data necessitates handling uncertainty in recognition from structure. Cooper attended a course on uncertain inference in the fall of 1986, for which he prepared a number of small but relevant papers, for orientation toward this goal. He has recently spent a larger fraction of his time working upon this problem, the results of which will be reported upon shortly.

B.2 A Probabilistic Approach to Low-Level Vision

Work has proceeded with a probabilistic approach to limited support boundary point detection. The approach uses the model developed in [Hueckel 1971] and [Canny 1986]--that is, step edges with uncorrelated Gaussian additive noise and linear blur. The algorithms implementing the approach can also handle other edge profiles and correlated noise. The algorithms return both probabilities and inverse probabilities, and have been shown to be superior to simple edge detectors such as those of Sobel and of Kirsch. These algorithms are documented in [Sher 1987a]. In the near future they will be tested against Nalwa's state-of-the-art edge detector [Nalwa and Binford 1986].

The detection algorithms have also been used for testing the theory for evidence combination developed in [Sher 1987b]. This work uses inverse probabilities (called likelihoods) for robust evidence combination. Inverse probabilities carry information about the fit of a model to the data thus can be used for added robustness. Work has shown that using the evidence combination to combine operators that assume differing levels of noise the combination achieves error rates as small as that of the best operator. Combining 9×9 and 5×5 detectors has resulted in detectors that have the noise resistance of the 9×9 detectors but the perturbation resistance of the 5×5 detectors (5×5 detectors are less sensitive to small perturbations from the model, such as having curved rather than straight edges). Combining different sized operators to get the strengths of both has long been an objective of computer vision [Marr 1982].

Sher's detectors have been tested using a set of graphics programs developed by Myra Van Inwegan. These programs generate images with shapes chosen at random with random intensities and positions. Van Inwegan also developed programs to add noise of specified distribution mean and standard deviation to images. An upcoming technical report describes this package.

All of this work was done using the C++ image processing environment. This environment makes it easy to implement simple image processing routines in a file format independent manner. This image processing environment is available in the public domain.

Sher has also consulted with Paul Chou on his work on information fusion for vision. Chou is using Sher's software as a first stage for his Markov random field work based on [Marroquin 1985] and documented in [Chou 1987].

A probabilistic approach also facilitates the low-level vision task of template matching. Template matching is used for object recognition. A template is a representation of the appearance of an object that is looked for in the scene. A probabilistic analysis of template matching yields algorithms to:

1) Translate the results of template matching into probabilities and inverse probabilities for the presence of the object.

2) Take the possibility of occluding objects into account with a Markov random field.

3) Weight a template in an optimal manner.

These techniques will be documented in Sher's forthcoming thesis.

Another profitable line of research involves deriving prior probabilities from user-provided models. As an example the prior probability of a boundary point can be deduced from the average size of objects in a scene.

B.3 Information Fusion for Multi-Modal Segmentation

Chou's thesis research addresses the problem of integrating the disparate sources of information available in low-level image computations to obtain scene properties of the image segments. He has identified the different characteristics of the available information and has proposed integration tools to utilize them.

In [Chou and Brown 1987a, b], a probabilistic approach to combining information from various sources for image segmentation has been proposed. In this approach, observable evidence and prior knowledge are separately modeled due to their distinct characteristics. Bodies of evidence are modeled as opinions provided by a set of early visual modules about individual image elements based on disparate sources of image observations. These opinions, represented as likelihood ratios with respect to a set of hypotheses about the image elements, are combined coherently and consistently through a hierarchically structured knowledge tree by propagating each opinion up and down the tree with simple computations. The combined opinions are shown, under some conditional independence assumptions, to be the joint likelihood ratios in Bayesian probability theory. Prior knowledge of spatial interactions of the image features is modeled with Markov Random Fields so that it can be characterized by a small set of parameters associated with the various configurations of local neighborhoods. *A posteriori* probability distributions of segmentations resulting from combining the prior knowledge and the available opinions following Bayes' rule are maintained incrementally and represented in a distributed fashion. These distributed representations could be used by several

estimation methods, such as the simulated annealing algorithm for MAP estimations [Geman and Geman 1984] and the Monte Carlo algorithm for MPM estimations [Marroquin et al. 1985], to produce statistically optimal estimations for segmentations under Bayesian decision rationale.

Preliminary experimental results, with synthetically generated images as input and a set of *likelihood edge detectors* [Sher 1987] to compute likelihood ratios, have shown several advantages of this approach:

1) Qualitative knowledge of the image features can be encoded adequately with the *a priori* probability distributions characterized by the local characteristics of Markov Random Fields.

2) Modules (experts) that know only a part of the set of hypotheses individually can be independently designed. Their opinions can be combined coherently.

3) Prior knowledge and observable information are integrated following Bayesian probability theory. The *a posteriori* probability distributions are constantly maintained to reflect the up-to-date knowledge.

4) Well-established statistical decision theories can easily be adopted to estimate segmentations.

5) The modularization of this approach simplifies the design and implementation of large low-level vision systems.

Chou [1987] has argued that stochastic estimation methods as well as some existing deterministic methods [Cohen and Cooper 1987] are inadequate for applications with computational constraints. In general, disparate sources of information might not be present at certain time and space. However, a segmenter should be able to provide higher-level processes reasonable segmentation estimations upon requests. With the above fusion mechanism, Chou has designed and implemented a deterministic estimation procedure that dynamically adjusts its estimations as new bodies of evidence arrive. Basically, this procedure maintains a priority queue; the image element with the least stable current estimation (under a stability measurement related to its neighboring estimations and external observations) is always the next to check. Like every deterministic local-minimization algorithm, this procedure does not guarantee to lead to the MAP estimator. However, its results are comparable to the stochastic estimation methods and usually superior to the existing deterministic method. An intuitive explanation for these promising results is that this procedure starts with a reasonable initial estimation and follows a path that is likely to converge to good results. Chou and Raman have implemented a simulation package to compare various estimation methods. This package and a set of experimental results will be described in a forthcoming technical report [Chou and Raman 1987].

B.4 Computer Vision on a Multiprocessor

B.4.1 Utilities and Benchmarks

Olson has been looking at software architectures for combining the output of independent low-level vision processes on the BBN Butterfly Multiprocessor. As a vehicle for studying these issues Olson implemented a two-dimensional image segmentor. The program iteratively splits regions taken from an active list until the list is empty; the choice of where to split is made by reconciling the recommendations of a user-supplied set of segmentation experts. The current implementation has only a single expert, a single-band version of the multiband histogram-based splitter used by Shafer and Kanade in the Phoenix system [Shafer and Kanade 1982]. Olson reported on his experiences with the segmentor at the DARPA Workshop on Blackboard Architectures for Image Understanding in June of 1986.

In support of the above work and in connection with Chris Brown's vision practicum course, Liudvikas Bukys and Olson adapted parts of the IFF/UBX image processing package to the Butterfly. As used at Rochester, IFF is an image file format, a standard filter-oriented style of writing vision applications, and a library of useful programs (filters, edge detectors, et cetera). Our efforts divided into three subprojects: a) Porting the IFF bit-oriented file packing and unpacking library to the Butterfly environment, b) providing an appropriate replacement for UNIX file system and pipes, and c) rewriting existing IFF utilities to take advantage of the Butterfly's capabilities. The first subproject was relatively simple, thanks to clean machine-independent design on the part of IFFs original authors. The only changes we made were to replace file accesses with access to TCP/IP connections to file server demons on remote machines. The second subproject was handled on the Butterfly end by a package (written by Olson). Olson joined several other members of the Rochester Vision Group in implementing the DARPA Image Understanding benchmark set [Simpson et al. 1986] on the Butterfly. For one benchmark Olson implemented a line-finding Hough Transform algorithm and compared it to a similar program he had written last year. This work is described in [Olson 1986b]. Olson also worked with Liud Bukys to adapt some of the BIFF utilities to meet the requirements of another of the DARPA benchmarks. He was able to show that for the benchmark tasks the Butterfly is almost completely CPU bound rather than communication bound. This work is described in a technical appendix to [Brown et al. 1986].

B.4.2 Concurrent Memory Allocation

In cooperation with Carla Ellis, Olson has been working on concurrent versions of the well-known first-fit memory allocation algorithm [Knuth 1968]. This work assumes a shared-memory machine supporting the fetch-and-add instruction [Gottlieb, Lubachevsky, and Rudolph 1983]; such machines include the BBN Butterfly, the NYU Ultracomputer [Gottlieb, Grishman et al. 1983] and the IBM RP3 [Pfister et al. 1985]. This work was motivated by our desire to improve on the solution of Stone [Stone 1982], which requires off-line storage proportional to the

number of blocks in the free storage list. We have designed a number of algorithms that trade overhead for concurrency in various ways. A description of the algorithms will appear in [Ellis and Olson 1987]. Stuart Friedberg provided valuable advice in the course of this work. Olson is implementing the algorithms on the Butterfly, and hopes ultimately to be able to evaluate their performance under various simulated load conditions.

B.4.3 Computational Models of Human Motion Perception

Olson has been studying the architecture of the human motion processing system. So far this has been mostly a matter of reviewing the literature rather than conducting new experiments, but it has led to some interesting conclusions. Following [Braddick 1974], Olson believes that the system can be divided into short- and long-range processes. However, it is clear from the nature of Braddick's experiments and the neurophysiological facts that the short-range process has a much greater spatial range than Braddick originally believed. Braddick set the limit of the short-range process at 15 min of arc. This value was accepted by Marr and Ullman [Marr 1980; Ullman 1979], and strongly affected their theories of motion processing. Olson prefers to identify the short-range process with receptive fields in striate visual cortex, and set the limit at one to two degrees of arc. Taken together with classical work on apparent motion [Kolers 1972], this change leads to the following picture. The short-range process treats motion as an abstract property of the image function. It is retinotopic, two-dimensional, and ignores any higher-level information about the scene. It can serve as the basis for segmentation (as in Braddick's experiments) but does not operate on segmented input. Its output is probably a noisy approximation to the optical flow field. Exactly what mathematical property of the image it measures is subject to debate. Olson favors the modified spatio-temporal energy formulation of [Adelson and Bergen 1986], but other energy formulations [Watson and Ahumada 1985; Adelson and Bergen 1985; Van Santen and Sperling 1985] or gradient-based methods [Fennema and Thompson 1979; Horn and Schunck 1981] are also reasonable. All of these methods compute more or less the same thing, so Olson does not regard the debate as crucial to his work.

The long-range process is vastly different. Its spatial range covers nearly the whole visual field, and it integrates information over time intervals as long as half a second. For the long-range process, motion is a property of high-level features (segments or objects). It recognizes the identity of objects over time and, in the special case of an apparent motion stimulus, chooses correspondences between features. Its choice is based on a complex metric involving plausibility of the speed and trajectory, figural match, salience, and even the expectations and desires of the observer.

B.5 Analyzing Massively Parallel Computation

Recently, researchers in Artificial Intelligence have been actively investigating various connectionist models of computation, also referred to as neural networks. Sara Porat frames her work in this area mostly as a connectionist model, defined by elementary processors that are similar to binary threshold units. Thus, we assume a

finite discrete space of states. This architecture has became one of the popular means of exploring the question of intelligence. Some recent works relate to the connectionist model as a model of computation and discuss its similarity to other non-uniform computational models, its computational power, and its complexity. We proceed in this direction and explore theoretic arguments that are natural in usual computational models, within this context of neural networks.

The model that is often studied is that of an asynchronous, symmetric network, where a global energy/goodness measure can be established and used to prove that the network totally stabilizes. This symmetry condition is somewhat unnatural for biological reasons, and moreover it precludes many computations that are biologically important. Certainly, any behavior that requires a loop, cycle or oscillation cannot be described by a monotonic goodness function.

In [Porat 1987] we discuss asymmetric networks, that might admit infinite activated computations. Within this framework, we define an operational semantics and analyze formally flow properties of some specific structured network with respect to a given specification (or correctness criterion) that characterizes the dynamics of an oscillator. We discuss the influence of a formal specification on the design of the network's structure, computational ability of its units, connections between them and rules of timing. We prove formally the behavioral correctness of some implementations, using a slightly different approach from that defined through the energy function, basically by proving logical assertions.

By shifting the discussion to asymmetric neural networks, it is natural to ask, for a given network, whether or not it stabilizes totally. We regard this property of stability as a major specification, while characterizing the behavior of a given network. This raises the importance of exploring the complexity of this decidable question. We prove the NP-hardness of this question under a synchronous activation rule, and similarly under a fair asynchronous rule. We also show that this problem is solvable in polynomial space. This investigation is original in this context of neural networks, and it motivates further research on other correctness assertions within this model.

This year, Porat attended the Foundations of Computer Science (FOCS) Conference in Toronto (October 1986), and was invited by the Computer Science Department at Carnegie Mellon University to give a seminar on the subject "Fairness in Models for Nondeterministic Computations" (November 1986).

B.6 References

Adelson, E.H., and J.R. Bergen. Spatiotemporal Energy Models for the Perception of Motion, *J. Opt. Soc. Am. A. 2*, 2, 1985.

Adelson, E.H., and J.R. Bergen. The Extraction of Spatio-temporal Energy in Human and Machine Vision, *Proc., IEEE Workshop on Motion Representation and Analysis*, Kiawah, SC, 1986.

Braddick, O.J. A Short-Range Process in Apparent Motion, *Vision Research 14*, 1974.

Brown, C.M., R. Fowler, T. LeBlanc, M. Scott, M. Srinivas, L. Bukys, J. Costanzo, L. Crowl, P. Dibble, N. Gafter, B. Marsh, T. Olson, and L. Sanchis. DARPA Parallel Architecture Benchmark Study, Butterfly Project Report 13, Computer Science Dept., Univ. Rochester, October 1986.

Canny, J. A Computational Approach to Edge Detection, *IEEE Trans. on Pattern Analysis and Machine Intelligence PAMI-8*, 6, 679-698, November 1986.

Chou, P.B. Multi-Modal Segmentation Using Markov Random Fields, to appear, *Proc., IJCAI-87*, Milano, Italy, August 1987.

Chou, P.B. Dynamic Multi-Modal Image Segmentation, internal documentation, Computer Science Dept., Univ. Rochester, March 1987.

Chou, P.B. and C.M. Brown. Multi-Modal Segmentation Using Markov Random Fields, *Proc., Darpa Image Understanding Workshop*, 663-670, Feb. 1987a.

Chou, P.B. and C.M. Brown. Probabilistic Information Fusion for Multi-Modal Image Segmentation, to appear, *Proc., IJCAI-87*, August 1987b.

Chou, P.B. and R. Raman. Relaxation Algorithms Based on Markov Random Fields, forthcoming Technical Report, Computer Science Dept., Univ. Rochester, 1987.

Cohen, F.S. and D.B. Cooper. Simple Parallel Hierarchical and Relaxation Algorithms for Segmenting Noncausal Markovian Random Fields, *IEEE Trans. Pattern Analysis and Machine Intell. PAMI-9*, 2, 195-219, March 1987.

Cooper, P.R. Order and Structure in Correspondence by Dynamic Programming, submitted, *International Journal of Computer Vision*, January 1987.

Cooper, P.R. and S.C. Hollbach. Parallel Recognition of Objects Comprised of Pure Structure, *Proc., DARPA IU Workshop*, Los Angeles, CA, February 1987; submitted, *AAAI-87*.

Cooper, P.R., D.E. Friedmann and S.A. Wood. The Automatic Generation of Digital Terrain Models from Satellite Images by Stereo, *36th Congress of the Int'l. Astronautical Federation*, Stockholm, Sweden, October 1985; to appear, *Acta Astronautica*.

Ellis, C.S. and T.J. Olson. Parallel First Fit Memory Allocation, to appear, *Proc., IEEE Int'l. Conf. on Parallel Processing*, 1987.

Fennema, C.L., and W.B. Thompson. Velocity Determination in Scenes Containing Several Moving Objects, *Computer Graphics and Image Processing 9*, 1979.

Geman, S. and D. Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images, *IEEE Trans. Pattern Analysis and Machine Intelligence PAMI-6*, 6, 1984.

Gottlieb, A., B.D. Lubachevsky, and L. Rudolph. Basic Techniques for the Efficient Coordination of Very Large Numbers of Cooperating Sequential Processors. *ACM Trans., Programming Languages and Systems 9*, 6, April 1983.

Gottlieb, A., R. Grishman, C.P. Kruskal, K.P. McAuliffe, L. Rudolph and M. Snir. The NYU Ultracomputer--Designing an MIMD Shared Memory Parallel Computer, *IEEE Trans. Computers 32*, 2, February 1983.

Hollbach, S.C. Tinker Toy Recognition from 2D Connectivity, TR 196, Computer Science Dept., Univ. Rochester, October 1986.

Horn, B.K.P. and B.G. Schunck. Determining Optical Flow, *Artificial Intelligence 17*, 1981.

Hueckel, M.H. An Operator Which Locates Edges in Digitized Pictures, *Journal of the Assoc. for Computing Machinery 18*, 1, 113-125, January 1971.

Kolers, P.A. *Aspects of Motion Perception*. New York: Pergamon Press, 1972.

Knuth, D.E. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Addison-Wesley, 1968.

Marr, D. *Vision*. San Francisco: W.H. Freeman, 1980.

Marroquin, J.L. Probabilistic Solution of Inverse Problems, Tech. Rept. 860, MIT Artificial Intelligence Laboratory, September 1985.

Marroquin, J.L., S. Mitter, and T. Poggio. Probabilistic Solution of Ill-Posed Problems in Computational Vision, *Proc., DARPA Image Understanding Workshop*, December 1985.

Nalwa, V.S. and T.O. Binford. *IEEE Trans. Pattern Analysis and Machine Intelligence PAMI-6*, 679-698, November 1986).

Olson, T.J. An Image Processing Package for the BBN Butterfly Parallel Processor, Butterfly Project Report 9, Computer Science Dept., Univ. Rochester, August 1986a.

Olson, T.J. Finding Lines with the Hough Transform on the BBN Butterfly Parallel Processor, Butterfly Project Report 10, Computer Science Dept., Univ. Rochester, August 1986b.

Olson, T.J., L. Bukys and C.M. Brown. Low-level Image Analysis on an MIMD Architecture, to appear, *Proc., First Int'l. Conf. on Computer Vision*, London. June 1987.

Pfister, G.F., W.C. Brantley, D.A. George, S.L. Harvey, W.J. Kleinfelder, K.P. McAuliffe, E.A. Melton, V.A. Norton and J. Weiss. The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture, *Proc., ICPP*, 1985.

Porat, S. Stability and Looping in Connectionist Models with Asymmetric Weights, TR 210, Computer Science Dept., U. Rochester, 1987.

Porat, S. and N. Francez. Fairness in Context-Free Grammars under Every Choice-Strategy, to appear, *Information and Computation*, 1987.

Shafer, S. and T. Kanade. Recursive Region Segmentation by Analysis of Histograms, *Proc., IEEE Int'l. Conf. on Acoustics, Speech, and Signal Processing*, Paris, France, May 1982.

Sher, D.B. Evidence Combination Based on Likelihood Generators, TR 192, Computer Science Dept., Univ. Rochester, January 1987a.

Sher, D.B. Advanced Likelihood Generators for Boundary Detection, TR 197, Computer Science Dept., Univ. Rochester, January 1987b.

Simpson, R., S. Squires, and A. Rosenfeld. Strategic Computing Vision Architecture Benchmarks, private communication, July 1986.

Stone, H.S. Parallel Memory Allocation using the FETCH-AND-ADD instruction, IBM T.J. Watson Research Center Tech. Rept. RC9674, November 1982.

Ullman, S. *The Interpretation of Visual Motion*. Cambridge, MA: MIT Press, 1979.

van Santen, J.P.H. and G. Sperling. Elaborated Reichardt Detectors, *J. Opt. Soc. Am. A* 2, 2, 1985.

Watson, A.B. and A.J. Ahumada, Jr. Model of Human Visual-Motion Sensing, *J. Opt. Soc. Am. A* 2, 2, 1985.

DARPA Parallel Architecture Benchmark Study

C. Brown, R. Fowler, T. LeBlanc, M. Scott, M. Srinivas, L. Bukys, J. Costanzo,
L. Crowl, P. Dibble, N. Gafter, B. Marsh, T. Olson, L. Sanchis

October 1986

Abstract

In intensive work over a four-week period in the summer of 1986, seven problems were studied and implemented on the Butterfly. The problems were inspired by various capabilities in computer vision, and were proposed as benchmarks for a DARPA workshop on parallel architectures. They were: convolution and zero-crossing detection for edges, edge tracking, connected component labeling, hough transform, three computational geometry problems (convex hull, voronoi diagram, and minimum spanning tree), three-dimensional visibility calculations, subgraph isomorphism and minimum cost path calculation. BPRs 10, 11, and 14 are detailed reports on three of the problems. BPR13 contains the conclusions of the study and writeups of the work not covered in other BPRs.

This work was supported in part by the Defense Advanced Research Projects Agency U.S. Army Topographic Labs under grant number DACA76-85-C-0001 and in part by the National Science Foundation under grant number DCR-8320136.

Table of Contents

Chapter One: Overview

Overview

Christopher Brown, Tom LeBlanc, Michael Scott
Computer Science Department
29 August 1986

1. Disclaimer

The University of Rochester's response to the DARPA Architecture Workshop Benchmark Study request was a three-week period of activity, commenced from a standing start with the arrival of the problem specifications (Chapter 2). During this time the researchers had to make difficult technical decisions very quickly and to implement and run experiments under severe time pressure. Often sub-optimal methods were chosen for practical reasons. Some of the work has led to internal technical reports, and much of the work is being followed up and will appear in more finished form elsewhere. The contents of this report represent a snapshot of work not currently written up elsewhere, as of our self-imposed deadline of 1 September 1986 (The Architecture Workshop was later rescheduled to mid-November 1986).

The contents of this report represent preliminary work, and should not be considered our best or final answers to the various levels of problems raised by the Benchmark Study.

2. The Study

Rochester's DARPA Architecture Workshop Benchmark Study is made up of several chapters, each written by an individual or a small group. This, Chapter 1, gives an overview of the work and the resulting conclusions. Chapter 2 is a formatted version of the original memo that gave the problem specifications.

The remainder of this document, Chapters 3-9, along with separate Computer Science Department *Butterfly Project Reports*, (numbers 10, 11, and 14) detail technical aspects of our work on individual problems. Generally there is one chapter per problem, except that we used the connected components algorithm (Problem 2, described in *BPR 11*) to do edge-following (Problem 1.c.) as well. Thus Chapter 3 gives results on edge-finding and zero-crossing detection, while Chapter 4 (*BPR 11*) discusses the work on edge-following and connected components. Chapte 5 is equivalent to *BPR 10* and Chapter 8 is equivalent to *BPR 14.*

3. The Effort

Over a three-week period, several students and faculty at the University of Rochester's Computer Science Department worked on the seven architecture

benchmarks proposed by Rosenfeld, Squires, and Simpson (Chapter 2). Because of the short time and limited personnel resources available, the results reported here should not be considered as our last word on any of the problems. We did, however, find the exercise to be stimulating and a good investment. Our report takes the form of this brief summary document and a collection of chapters written by individuals and small groups who addressed individual problems.

Those directly involved in the effort were two staff members, five faculty members, and six graduate students varying from pre-first-year to third year in the areas of artificial intelligence, systems, and theory. The concentration of work was relatively intense, varying from approximately 20% effort to 75% effort per person over the three week

Rochester's place in the Strategic Computing program is to investigate and build programming environments for parallel vision. With this charter, we felt that the more benchmark implementations we could build the better. Further, in the area of programming advanced parallel architectures, often interesting software engineering must be done to improve implementations in the face of performance facts. We believe that theoretical or simulated results, while safer to propound, are of limited interest. Beyond our desire to get programs running, our goals were diverse.

(1) The primary goal is to evaluate the Butterfly Parallel Processor architecture and its existing software resources.

(2) Some of us wanted to use and test utilities we had already developed (e.g. the BIFF utilities used for the edge-finding task and the UNION-FIND package used for connected component labelling.)

(3) Some wanted to code applications in recently-implemented parallel languages and program libraries (e.g. LYNX was used in the triangle visibility task, and the Structured Message Passing library was used in the shortest path problem).

(4) Some wanted to modify and extend existing projects (e.g. the undirected edge-detector extension for the Hough transform task. Another example was an experimental modification of a clustering program to do the minimum spanning tree task -- that work is not reported here.)

(5) Some wanted to explore the mapping of current parallel algorithms from the theoretical literature onto parallel architectures, and to open research avenues in this direction (e.g. the subgraph isomorphism task, which has already generated interesting new scientific results, and the computational geometry tasks).

There was little problem in implementing most of the problems. All told, four programming environments were used:

(1) C and raw Chrysalis (the Butterfly operating system)

(2) The Uniform System of BBN

(3) Structured Message Passing (developed at Rochester)

(4) LYNX (ported to the Butterfly at Rochester).

The programmers ranged from naive first-time users of the Butterfly to highly experienced and sophisticated programmers who could (and did) modify system internals to improve performance.

4. The Problems

The original problem statements appear in the next chapter. Detailed write-ups of our approach to and results on the problems follow in separate chapters. The problem statements were followed as closely as made sense given the scientific goals of the study. For example, in the triangle visibility problem, floating point was not used because the inefficient software implementation of floating point would distort the interesting statistics. (The Butterfly does in fact need the Floating Point Platform upgrade if it is to be useful in serious scientific computing.) In the convex hull problem we went to a larger-than-specified problem size because of results with sequential implementations, and in the graph isomorphism problem we used a smaller problem size than specified for technical reasons. An ambiguity in the shortest path problem statement was interpreted in a way that was *not* advantageous to the Butterfly architecture but seemed to be indicated by the "hint" in the problem statement, and which was more practical given the time constraints. Wherever we have changed a problem specification we have tried to explain why, and tried to indicate responsibly what the consequences of literal interpretation would have been.

We chose the Butterfly several years ago because, among other things, its hardware architecture imposed the *least* constraint on the abstract models of computation it supported. Thus mapping problems onto the Butterfly is a doubly interesting exercise. There is a theoretical phase in which a good algorithm (using one or another abstract computational model) is chosen and the abstract model is matched with a Butterfly programming environment. Then there is an engineering phase in which the implemention is built and made efficient. The best results occur when both these phases are done well. In this set of problems sometimes the first phase went well but the second phase was not attempted (as in the geometry problems we did not implement) or needs more work (as in the triangle visibility problem). Also there were some cases in which the first phase was given short shrift because it looked like a research problem (e.g. the subgraph isomorphism problem), but the second phase was done very stylishly.

The computational domain of the benchmark was not one that could fully take advantage of the Butterfly's MIMD architecture. One computational aspect lacking in the benchmark problems is the case of a cooperating set of independent programs, such as occurs in client-server models. The benchmark tested performance (of programmers, languages, architectures, operating systems, programming environments) on single algorithms solving easily-stated problems. This limitation is worth noting since, in advanced systems, cooperation and

communication between basically independent processes will be important. Also the benchmark problems were small compared to a working AI system. Another set of benchmark problems to illuminate these issues could be proposed and might include construction of a file system, or a system in which results of disparate, asynchronously computed results are merged.

Within its limited perspective, the benchmark did comprise a diverse and challenging set of problems, and allowed us to reach several conclusions. For details on the technical approaches, performance, and individual conclusions, see the following chapters. The next section gives some highlights of our observations.

5. Observations

It is difficult to boil down the diversity of our results into a small set of out-of-context conclusions. Nevertheless, the following observations seem safe.

(1) During the last year, advances made at BBN and at the University of Rochester have made the Butterfly much easier to program under several complementary models of computation. A programmer starting with only a knowledge of standard sequential programming can now produce parallel programs (in the Uniform System or Structured Message Passing) in a day or two. Alternatively, knowing a modern language like Ada would make learning LYNX, and subsequent Butterfly programming, quite easy.

(2) The Butterfly can be efficiently (as well as easily) programmed using several "virtual architectures" (models of parallel computation).

(3) The Butterfly architecture can implement a wide variety of abstract parallel models of computation. Further, the combination of significant local memory and quickly accessible "shared" memory gives the capability for several complementary types of parallelism working together. While programming environments that emphasize one or another parallel model are available now, a single environment that gives the programmer access to a mix of computational models is not. The subgraph isomorphism problem illustrates one case in which a mix would have been useful. At Rochester the PSYCHE project has the goal of providing unified support for a variety of parallel computation models, including both shared memory and message-passsing.

(4) For serious work in the area of scientific computation covered in the benchmark, and probably for general programs, the new Butterfly Floating Point Platform is a necessity. Both floating point operations and integer multiplies are a serious bottleneck (see the Hough Transform Problem).

(5) Microcode support for debugging and performance monitoring would be a significant improvement. There would be considerable payoff in a small microcode fix to provide 32-bit atomic operations. One specific (and easy) upgrade would be microcode hooks to allow logging atomic operations. This facility would allow a reliable record of the order that processes enqueued entries on dual queues.

(6) Memory management is a serious problem. The scarcity of Segment Attribute Registers makes their management a major concern. The inability of the 68000 to do demand paging is also awkward (here again the problem is solved by the Floating Point Platform upgrade). Very large memory objects (larger than one physical memory) are an interesting issue that a few groups are working on -- some benchmark problems (e.g. shortest path) expose the desirability of a clean solution for the large object problem.

(7) A switch that supported simultaneous communication with several destinations would improve the implementation of broadcast or multicast communication used in many algorithms. A combining switch might reduce memory contention, but its efficacy is a research issue.

(8) The Uniform System really provides a global shared name space, not a shared memory. To achieve good speedup of parallel algorithms, local memory must be used to avoid memory contention. Even knowing the standard tricks is not enough to guarantee good performance. The Hough Transform chapter provides an interesting example evolution of program ameliorations. A "shared memory" (as in the planned Monarch) would seem to support Uniform System style programming better. However, it is doubtful that remote memory can ever be made as fast as local memory, and so the local-global question cannot be avoided. A very fast block-transfer capability would improve matters in the current architecture, and would not close off any options in the computational models the Butterfly would support. However, the block-transfer fix does not address the local-global conflict at the conceptual level. Similarly, the fast-switch "shared-memory" does not solve the local-global conflict at the technical level. What is needed perhaps is continued diversification of the abstract models of computation available and in the programming environments that support them.

(9) Amdahl's law is important, and any serial program behavior has serious adverse consequences in speedup performance. Such serialization sometimes hides in the system software and special effort (and talent) are required to avoid or fix it (e.g. the parallel memory allocation modification introduced in the convex hull implementation). The timings shown in Chapter 4 (BPR 11, connected components) and Chapter 7 (triangle visibility) are revealing. Systems code to allocate memory and replicate data dominates times if it is not parallel.

(10) Software to support (efficiently) many more processes than processors on the Butterfly would make implementing a great many important algorithms easier. There are many algorithms in which a process is dynamically allocated to each problem object (e.g. the nodes of a graph), for large numbers of objects. The Uniform System does not answer because it is unable to do process synchronization: processes cannot be blocked, unscheduled, awakened, etc. One reasonable response to this need would be a programming environment such as Concurrent Euclid (Ada would be usable

but not as good), with monitors or a similar means of concurrency control/encapsulation. The actual composition of the environment is a research issue, but it may be necessary to have something like a full blown object-oriented system in which tasks are represented as first class entities encapsulating data, code, and process.

6. Concluding Remarks

The Benchmark Study was a stimulating exercise that served several purposes at Rochester. It has led to new software utilities, to new understandings of Butterfly strengths and weaknesses, to applications for new programming environments developed at Rochester, and to new research avenues in parallel algorithm theory and development. It has encouraged us that our work in building programming environments for the Butterfly has been effective.

Several of the benchmark problems (e.g. the geometry problems) were useful but uncomfortable because they underlined current weak points in the programming systems we have. The graph algorithms need a high degree of cheap (i.e. not SMP or LYNX) parallelism that is independent (i.e. not US-style) -- thus they exposed fruitful areas for future research. We welcome such problems. Our goal is to get the most out of the flexible MIMD architectures of the future, and "counterexamples" to current solutions are always interesting and important. We believe that one of the most promising and important research areas centers around the goal of a single programming environment that can take advantage of the potential for several sorts of parallelism in tightly-coupled MIMD computers, and we are now working actively in in that area.

We believe that much can and will be gained by continuing with the method we have been pursuing at Rochester -- a symbiosis of theoretical, systems, and applications research. We shall continue to build systems for internal and external use that incorporate our theoretical insights and meet the needs of applications. With the basic research underlying the systems, and with the systems as tools, we and others will move forward toward understanding and controlling state-of-the-art parallel programming systems.

Chapter Two: Problem Specifications

DRAFT

MEMO TO: Designers of architectures for image understanding (IU)

FROM: Azriel Rosenfeld, Bob Simpson, Steve Squires

SUBJECT: New architectures for IU

DARPA plans to hold a workshop during the week of Sep- tember 8 in McLean, Virginia to discuss what the next steps should be in developing IU architectures that could be available to researchers by the 1990's.

A lot is known about architectures for low-level vision, but we need to move toward systems that can handle the total vision problem, including both the low- and high- level ends as well as the interface between the two.

Appended to this memo is a set of "benchmark" IU problems. We have tried to define them as precisely as pos- sible, so as to make it possible to predict how a given sys- tem would perform on them. (We have provided some refer- ences to the relevant literature for your convenience.)

You are invited to make such predictions for your (existing or proposed) systems, and to prepare a short paper documenting the results. This paper should be sent to us for distribution to the Workshop attendees by mid August, so everyone will have a chance to evaluate the results and dis- cuss them at the Workshop. If your system is not very effi- cient at some of the tasks, you may wish to indicate how you would improve or augment it to make it more efficient.

We look forward to hearing from you and to seeing you at the Workshop.

Appendix: IU benchmarks

(1) Edge detection
In this task, assume that the input is an 8-bit digital image of size 512 x 512 pixels.

a) Convolve the image with an 11 x 11 sampled "Laplacian" operator [1]. (Results within 5 pixels of the image border can be ignored.)

b) Detect zero-crossings of the output of the operation, i.e. pixels at which the output is positive but which have neighbors where the output is negative.

c) Such pixels lie on the borders of regions where the Laplacian is positive. Output sequences of the coordinates of these pixels that lie along the borders (On border following see [2], Section 11.2.2.)

(2) Connected component labeling
Here the input is a 1-bit digital image of size 512 x 512 pixels. The output is a 512 x 512 array of nonnegative integers in which

a) pixels that were 0's in the input image have value 0

b) pixels that were 1's in the input image have positive values; two such pixels have the same value if and only if they belong to the same connected component of 1's in the input image.

On connected component labeling see [2], Section 11.3.1.)

(3) Hough transform
The input is a 1-bit digital image of size 512 x 512. Assume that the origin (0,0) image is at the lower left-hand corner of the image, with the x-axis along the bottom row. The output is a 180 x 512 array of nonnegative integers constructed as follows: For each pixel (x,y) having value 1 in the input image, and each i, 0 << i << 180, add 1 to the output image in position (i,j), where j is the perpendicular distance (rounded to the nearest integer) from (0,0) to the line through (x,y) making angle i-degrees with the x-axis (measured counterclockwise). (This output is a type of Hough transform; if the input image has many collinear 1's, they will give rise to a high-valued peak in the output image. On Hough transforms see [2], Section 10.3.3.)

(4) Geometrical constructions
The input is a set S of 1000 real coordinate pairs, defining a set of 1000 points in the plane, selected at random, with each coordinate in the range [0,1000]. Several outputs are required.

a) An ordered list of the pairs that lie on the boundary of the convex hull of S, in sequence around the boundary. (On convex hulls see [3], Chapters 3-4.)

b) The Voronoi diagram of S, defined by the set of coordinates of its vertices, the set of pairs of vertices that are joined by edges, and the set of

rays emanating from vertices and not terminating at another vertex. (On Voronoi diagrams see [3], Section 5.5.)

c) The minimal spanning tree of S, defined by the set of pairs of points of S that are joined by edges of the tree. (On minimal spanning trees see [3], Section 6.1.)

(5) Visibility
The input is a set of 1000 triples of triples of real coordinates ((r,s,t),(u,v,w),(x,y,x)), defining 1000 opaque triangles in three-dimensional space, selected at random with each coordinate in the range [0,1000]. The output is a list of vertices of the triangles that are visible from (0,0,0).

(6) Graph matching
The input is a graph G having 100 vertices, each joined by an edge to 10 other vertices selected at random, and another graph H having 30 vertices, each joined by an edge to 3 other vertices selected at random. The output is a list of the occurrences of (an isomorphic image of) H as a subgraph of G. As a variation on this task, suppose the vertices (and edges) of G and H have real-valued labels in some bounded range; then the output is that occurrence (if any) of H as a subgraph of G for which the sum of the absolute differences between corresponding pairs of labels is a minimum.

(7) Minimum-cost path
The input is a graph G having 1000 vertices, each joined by an edge to 100 other vertices selected at random, and where each edge has a nonnegative real- valued weight in some bounded range. Given two vertices P,Q of G, the problem is to find a path from P to Q along which the sum of the weights is minimum. (Dynamic programming may be used, if desired.)

References

(1) R.M. Haralick, Digital step edges from zero crossings of second directional derivatives, *IEEE Transactions on Pattern Analysis and Machine Intelligence 6*, 1984, 58-68.

(2) A Rosenfeld and A.C. Kak, *Digital Picture Processing* (second edition), Academic Press, New York, 1982.

(3) F.P. Preparata and M.I. Shamos, *Computational Geometry - An Introduction*, Springer, New York, 1985.

Chapter Three: Edge Finding and Zero-Crossing Detection

Task One : Edge Detection

Thomas J. Olson

1. Introduction

The task is to detect edges in an eight-bit digital image of size 512 x 512 pixels. It is divided into three steps : convolution with an 11 x 11 Laplacian-of-Gaussian operator, zero crossing detection, and chain encoding sequences of connected zero crossings. In these experiments steps a) and b) were handled using image processing utility functions from the Butterfly IFF (BIFF) image processing library [3]. Step c) was performed by a special purpose routine adapted from a connected component labelling function. The test image was a densely textured natural scene in which approximately 25% of the pixels in the zero crossing image were ones. Our conclusions, briefly, are that for the 119-node Butterfly

a) convolution takes 3.48 seconds,

b) zero crossing detection takes 0.16 seconds, and

c) finding chain codes for lines takes 1.47 seconds for this image.

These times are for computational kernels; they do not include time to load the image, allocate memory for the output, et cetera. The sections that follow present implementation details and benchmarks for the first two steps. The third is described in the attached Butterfly Project Report [2].

2. Convolution

The convolution was performed by *convolve()*, the library version of the BIFF utility **iffconvolve**. *Convolve()* uses the Uniform System library [1] to perform a parallel FOR loop over rows of the output image. Each process created in this way does a sequential loop over rows of the mask; for each mask row it makes local copies of the appropriate mask and image rows and then does a linear convolution of the copied rows into a local array of 32-bit accumulators. Finally, it divides each accumulator by the user-supplied divisor and copies it to the output image.

It is easy to see that in order to produce an output, *convolve()* must perform $(502^2)(11^2) = 30{,}492{,}484$ multiplications. (The first term involves 502 rather than 512 because we ignore outputs within five pixels of the border.) Because it does so many multiplications, the execution time of *convolve()* is dominated by the 68000's multiplication time. Unfortunately the current Butterfly C compiler generates a call to an integer (32-bit) multiply routine even when the arguments are both shorts. Figure 1 shows timings and speedup curves for four versions of iffconvolve running an 11 by 11 mask on a 512 by 512 image. The first is coded in standard optimized C. The second replaces the multiplication in the innermost loop with a call to an assembly language

short multiply routine. For the third, we edited the compiler's assembly language output to replace the subroutine call with an in-line short multiply instruction. This is the version normally kept in the BIFF library. In the last version we replaced the multiply instruction with an addition. This gives an incorrect output, but indicates the sort of performance we might expect from a Butterfly with a fast full-parallel multiplier.

The *convolve()* routine is written to be as general as possible and therefor does not take advantage of some features of the problem as stated. First, the 11 by 11 mask to be used is known at compile time. This makes it possible to avoid copying the mask rows into local memory at execution time. The copy operation is quite fast, so we would expect the principal effect of this change to be a reduction in memory contention. The speedup curves of Figure 1 indicate that memory contention is not a serious problem for *convolve()*, so the net effect would be minor. Second, the mask is symmetrical. By factoring appropriately, the number of multiplies that the convolver must do can be cut almost in half. For example, a process working on input rows 0 through 10 would add row 10 to row 0, row 9 to row 1 et cetera, and then convolve rows 0 through 5 with rows 0 through 5 of the mask. Figure 2 shows the effect of these optimizations on the standard and simulated fast multiply versions of *convolve()*.

It should be noted that if we are willing to accept an approximation to the laplacian of a gaussian, we can speed the computation up substantially. Since gaussian masks are x-y separable, we can reduce convolution with an 11x11 mask to two convolutions with 11x1 masks. We can take advantage of symmetry as before, so that for each convolution we do 6 multiplies and 11 adds per pixel. The cost of an 11x11 gaussian convolution then becomes a mere 3,084,288 multiplies and 5,654,528 additions. We can compute the laplacian of the result by convolving with a 3x3 laplacian approximator. However, this method gives a relatively poor approximation to the truth. Better, though slightly more expensive, is to use the Difference of Gaussian (DOG) approximation, which requires two 11x11 convolutions followed by a pointwise difference. We have not benchmarked this method, but expect that it would reduce execution times by at least a factor of three (to about 1.1 seconds) based on the relative numbers of operations.

3. Zero Crossing Detection

For zero crossing detection we use the BIFF utility *zerocr()*. *Zerocr()* is written as a parallel FOR loop over output scan lines. Each row process reads in the corresponding input row and its two neighbors (the top and bottom rows are handled specially). For every positive pixel in the input row it examines the eight neighbors and stores a one in a local array if any of them is negative - otherwise it stores a zero. Finally it copies the local array into the output array. Timings are shown in Figure 3.

References

1. BBN Laboratories, The Uniform System Approach To Programming the Butterfly Parallel Processor, Version 1, Oct 1985.
2. L. Bukys, Connected Component Labelling and Border Following on the BBN Butterfly Parallel Processor, Butterfly Project Report 11, University of Rochester, Computer Science Department, Aug 1986.
3. T. J. Olson, An Image Processing Package for the BBN Butterfly Parallel Processor, Butterfly Project Report 9, University of Rochester, Computer Science Department, Aug 1986.

Figure 1 : Four Versions of lffconvolve

convolving 11x11 delsqg with 512x512 natural image
run times in seconds

procs	standard C	short mpy subroutine	short mpy in line	simulated fast mpy
4	617.52	318.68	153.53	121.11
8	313.66	159.35	78.00	61.53
16	156.84	80.95	39.00	30.77
32	78.44	40.49	19.52	15.40
64	39.23	20.26	9.77	7.71
119	24.52	12.66	6.11	4.82

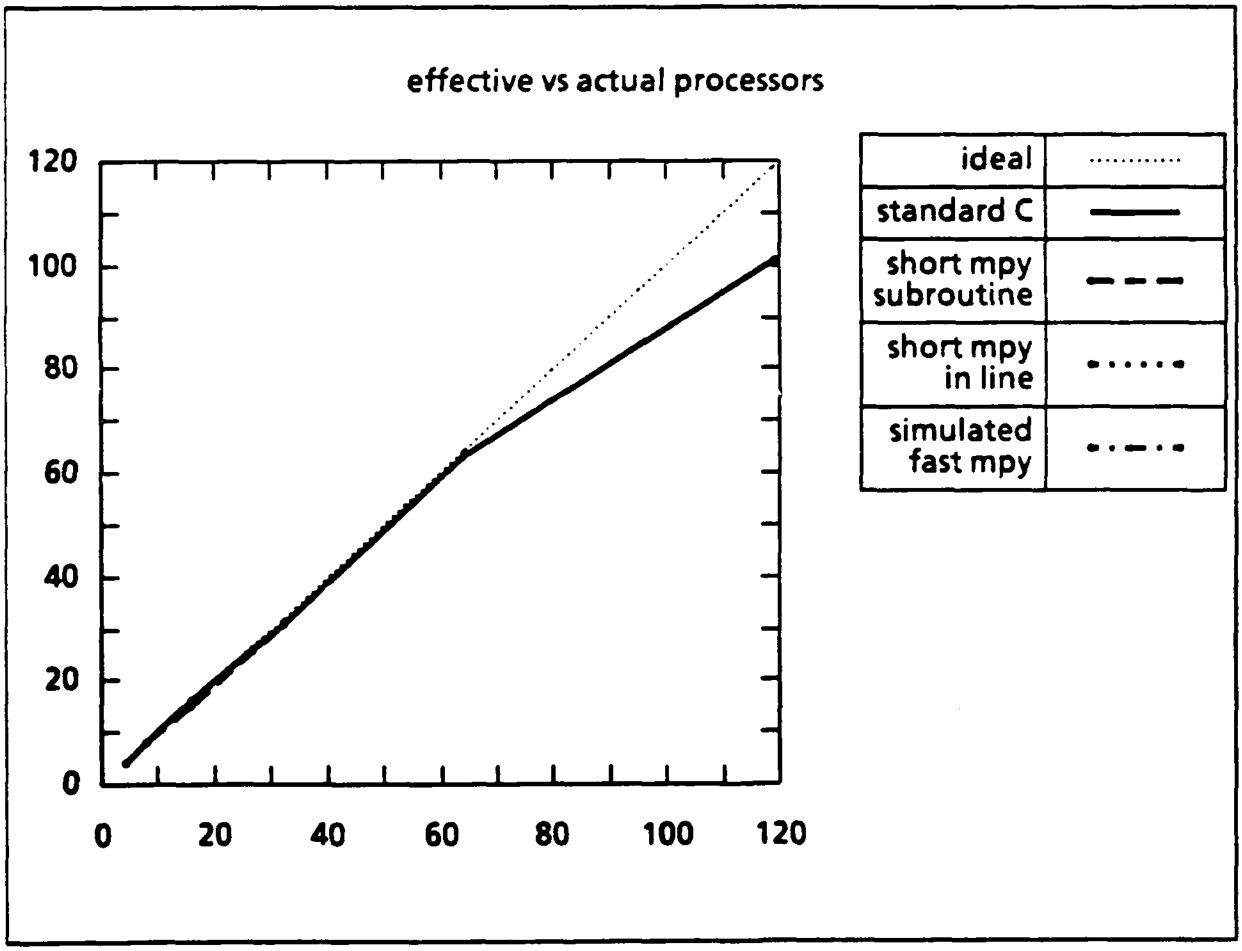


Figure 2 : Optimized lffconvolve

convolving 11x11 delsqg with 512x512 natural image
taking advantage of mask properties
run times in seconds

procs	standard short mpy in line	standard simulated fast mpy	optimized short mpy in line	optimized simulated fast mpy
4	153.53	121.11	87.27	68.25
8	78.00	61.53	43.58	34.04
16	39.00	30.77	22.14	17.30
32	19.52	15.40	11.08	8.66
64	9.77	7.71	5.56	4.35
119	6.11	4.82	3.48	2.72

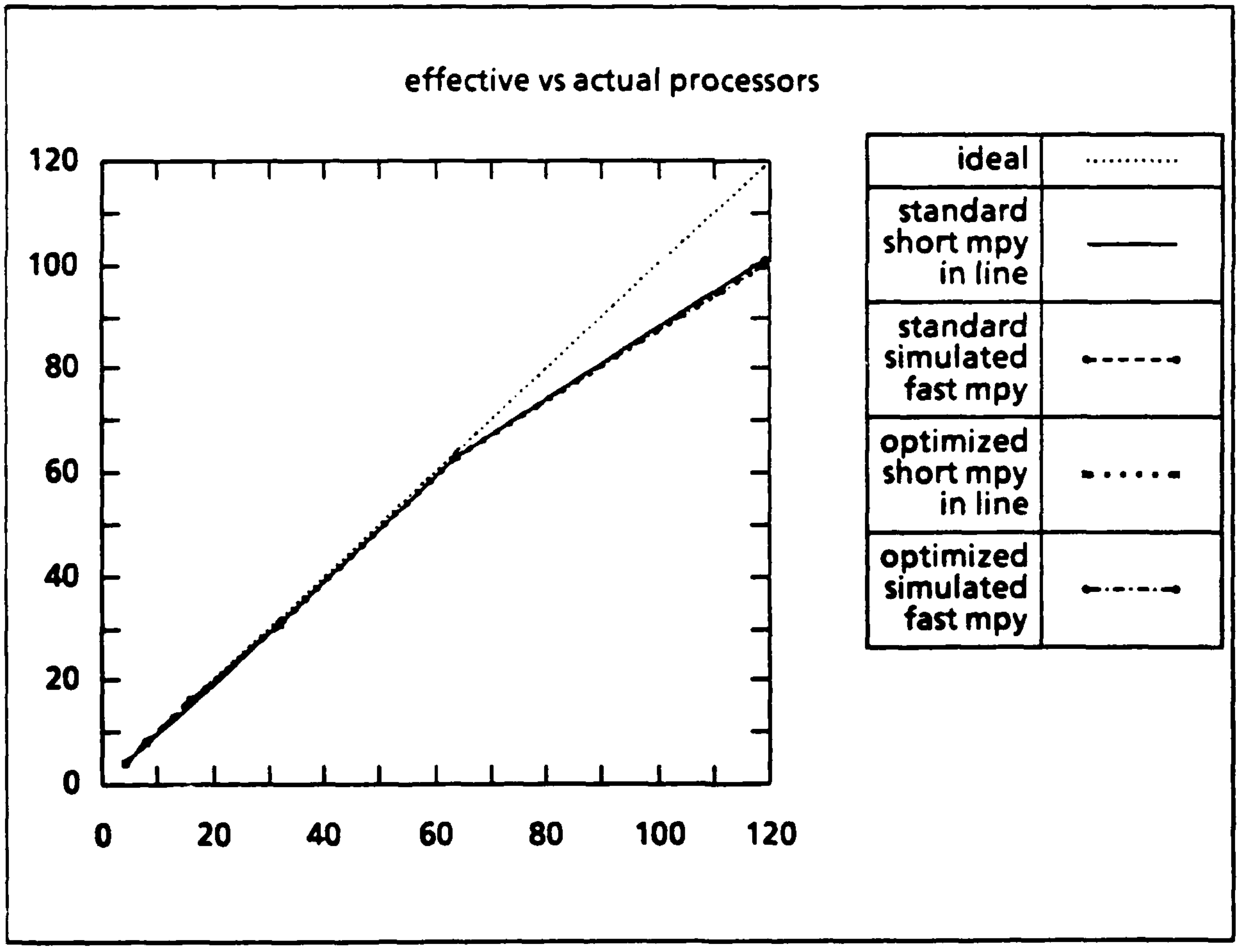
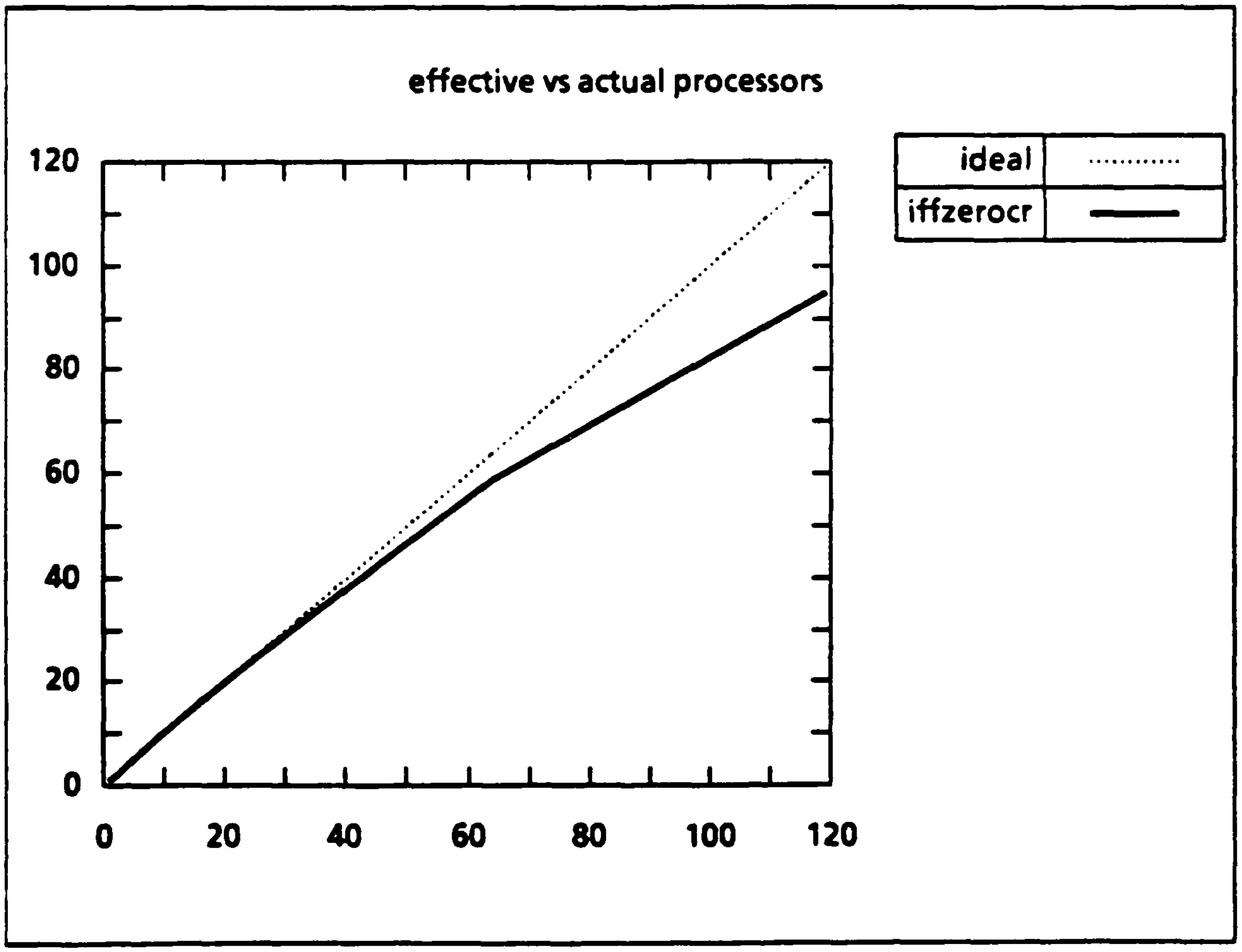


Figure 3 : zero crossing detection

run times in seconds

procs	iffzerocr
1	15.77
2	7.85
4	3.92
8	1.96
16	0.99
32	0.51
64	0.26
119	0.16



Efficient Convolution with Symmetric Masks

Tom Olson

The following amelioration to the inner loop of the 11x11 Laplacian convolution approximately halved the time needed for this portion of the benchmark, from 6.11 seconds to 3.48 seconds.

In the basic convolution multiply and add stage, every point (x, y) in the output can be computed by the expression

```
for row = 0 to 10
 for col = 0 to 10
  tmp = tmp + mask(row, col)*image(y+row, x + col)
```

which requires 121 adds and multiplies. We can use the symmetry properties of the mask to reduce the work. First, rewriting the above gives

```
for col = 0 to 10 tmp = tmp + mask(5, col)*image(y+5, x+col)
 for row = 0 to 4
  for col = 0 to 10
   tmp = tmp +
       mask(row, col)*image(y+row, x+col) +
       mask(10-row, col)*image(y+10-row, x+col);
```

Since mask(10-row, col) == mask(row, col), we can write this as

```
for col = 0 to 10 tmp = tmp + mask(5, col)*image(y+5, x+col)
 for row = 0 to 4
  for col = 0 to 10
   tmp = tmp +
       mask(row, col)*(image(y+row, x+col) + image(y+10-row,
x+col))
```

which takes the same number of adds but only 66 multiplies. We can do even better by realizing that we're going to do this at every point along a row. That means we can precompute the image row sums once and for all. That is, to compute row y in output, sum rows y and y+10, y+1 and y+9, ..., y+4 and y+6, and include row y+5 to get a 6 row by n column matrix. Then simply convolve with the bottom six rows of the mask.

In summary, the standard convolution for a 512x512 image and 11x11 mask, ignoring dropping edge outputs is
502x502x11x11 = 30,492,484 mpys and adds.

Using the symmetry of the mask, the code above reduces the counts to
502x512x5 = 1,285,120 adds to make the 502 6x512 matrices, plus

502x502x11x6 = 16,632,264 mpys and adds to do the individual convolutions.

So multiplies are reduced by almost half.

We can take further advantage of the available symmetries by folding the 6x11 mask we use in the implementation above around the middle column. This cuts the number of multiplies to 502x502x6x6 = 98,072,144. Similarly, by folding the 6x6 mask around its diagonal we can reduce the total number of multiplies to 21 per mask, giving 502x502x21 = 5,292,084 for the total. Unfortunately the number of additions stays constant at about 16M, and the loops and indexing become complex, so that it is not clear that these refinements will actually improve execution times. These techniques are applicable to any rotationally symmetric mask, so if they do prove worthwhile we will probably put a special convolution routine into the BIFF library for rotationally symmetric masks.

Chapter Four: Connected Component Labeling

see: Butterfly Project Report 11

Chapter Five: Hough Transformation

see: Butterfly Project Report 10

Chapter Six: Geometrical Constructions

Geometry Problems

Robert J. Fowler and Neal Gafter
August 21, 1986

1 Introduction.

The approach that we took in investigating the suitability of the Butterfly architecture for the geometric problems was to attempt the parallelization of good sequential algorithms for those same problems. Thorough discussions of such sequential algorithms for computational geometry may be found in [Mel84] and [PS86]. In [ACG*85] Aggarwal *et al* sketched some parallel algorithms for computational geometry derived from these sequential algorithms. Problems they addressed included the computation of two-dimensional convex hulls, Voronoi diagrams, and other problems. The model of computation that they used is the concurrent-read, exclusive-write (CREW) variant of the PRAM model of computation. The methods used by Aggarwal *et al* (at least for convex hull and the Voronoi diagram) are the parallelization of optimal sequential algorithms. We use a similar approach, but directed not towards the theoretically interesting questions of optimality in the asymptotic sense and of membership in well known complexity classes (e.g. NC), rather towards achieving good performance when implemented on the Butterfly. In particular, we are using these problems as an opportunity to explore how to map algorithms designed for abstract models of parallel computation onto a physically realized parallel architecture.

2 Salient Aspects of the Butterfly Architecture.

Each node on the Butterfly consists of a processor and a megabyte of memory local to that node. In addition, the nodes are connected with a "butterfly" (hence the name) interconnection network that allows each node to access the memory of the others. The interconnection network resolves contending attempts to access each memory module by serializing those attempts. Because of the large granularity of the memory modules this "hidden" serialization of contending access attempts can be a major problem with the approach of attempting to adapt PRAM algorithms that assume the possibility of very fine grained parallelism. We believe that the investigation of data structures that can be shared with low contention among a reasonable number of processors to achieve medium scale parallelism on such a machine is an area for potentially fruitful research.

The architecture seen by an application programmer is not determined solely by the underlying hardware, rather by a combination of hardware and the software architecture of the operating system cum programming environment. The latter can have as much or more of an effect as the former on the successful implementation of an algorithm on a particular machine. The quality of the programming environment affects both the ease of implementation as well as how well the underlying machine is used [Sny86].

Of the programming environments currently available on the Butterfly we chose the Uniform System because it most closely resembles the PRAM model. In the course of this exerecise we encountered the following specific problematic aspects of the Uniform System:

- The Uniform System appears to have been designed to be used in a style in which memory is statically allocated when the application is initialized and in which there is a small number of generators that spawn a large number of tasks. In contrast, the geometric problems naturally seem to fit into a style that uses dynamic memory allocation and in which tasks are spawned dynamically a few at a time as a program executes its recursive algorithm. There appear to be substantial penalties for using this latter style in the Uniform System.

- The geometric problems all involve the construction of a graph with some specified properties from a set of input points. An efficient parallel program to solve such problems must have efficient parallel implementations of the abstract data types *set* and *graph*. A natural representation of a graph is as some form of dynamic list structure. One consequence of this is that either the system or the application program should provide efficient parallel memory management. The global memory management provided by the Uniform System is in fact sequential. Thus, even a program that appears to be parallel can in fact be serialized by system code. We provided our own parallel memory management, but this illustrates how easy it is for implicit serialization to be introduced by the programming environment.

- Another consequence of using dynamic list data structures is the need to provide concurrency control for the elements. It is possible to do concurrency control in the Uniform System but it is awkward. The natural way of doing this is to incorporate the concurrency control mechanism in the programming language.

- The assignment of processors to tasks must be made more efficient and flexible. Task startup can introduce a substantial amount of overhead that can wipe out the benefits of fine and medium grain parallelism. In addition, we discovered that the implementation of task scheduling task allocation scheme can force a processor to be idle when it is not logically required by the application to be so and there is useful work it could do.

These factors contribute to the difficulty of using the Butterfly hardware architecture effectively. This illustrates the need for improved parallel programming environments as well as the need for those environments to provide the programmer with an accurate and detailed enough model of computation to guide intelligent choices in implementation.

Because our interest in these exercises is the investigation of the problem of mapping abstract algorithms onto the Butterfly we emphasized general implementations rather than attempting to tune the programs to exploit specific details of the problem statement(s). Thus, we are at least as interested in very large numbers of points distributed in arbitrary regions as we are in small numbers of points distributed uniformly in a square.

3 An Abstract Convex Hull Algorithm.

We concentrated our efforts on understanding the design and implementation of a parallel two-dimensional convex hull program. Most of the issues that would arise in the implementation of programs to solve the other problems appear in the convex hull problem and. in the limited time available, we deemed it more important to understand one implementation than to dilute our efforts by attempting "quick and dirty" implementations of all of the geometric problems.

Our approach is similar to that proposed by Aggarwal *et al*, but is an attempt to exploit the bounded, medium-grained parallelism found on a Butterfly. It is a parallelization of the Quickhull [PS86] algorithm. Our parallel implementation has the following steps:

1. Select the maximum and minimum elements along some direction using as much parallelism as is effectively available. We assume that there are N points and that we can use P processors effectively. Each processor is given a subset of approximately N/P elements of which it finds the maximum and minimum elements. The global maximum and minimum is computed from the subset values using the usual PRAM trick of combining the subset extrema using binary trees. The useful (non-overhead) part of this step requires time proportional to $N/P + logP$.

2. If the initial points are A and B, then the initial approximation to the hull is the ordered list of the directed line segments AB and BA. The remaining points are partitioned into two sets. one above AB and the other above BA. This is done in parallel, with each processor working on a subset of the input. To allow for parallel partitioning, a set of points can be represented as a tree of variable length arrays (sub-buckets) of points. The partitioning is done by having each processor copy its part of the input local memory using a block transfer. It then scans it's input sequentially while locally building its part of the output set consisting of those points above the specified line. The other points are discarded from the set because they can not contribute to the hull at this location. The sub-sets from each process are merged in parallel into the output trees. The reason for using a tree rather than a simple linked list is to allow for

efficient parallel access and to allow each internal node of the tree to keep a count of the points that it contains. As each sub-set is constructed the point furthest from the line is found. The local extrema are also combined in the binary tree to find the global extremem. This point is on the convex hull. The time for one of these steps is proportional to $N/P + \log P$.

3. At any time the current approximation to the hull is kept as a doubly-linked list of points. All of the as yet unknown points on the hull are outside this approximation so the points within it can be discarded. Furthermore, any point that can possibly be added to the hull between a pair of points in the current approximation must be above the line of support through them. When a newly found hull point is added to the list it replaces one of the line segments on the approximation with two others. A new task is created for each of these. Each task takes as its input a new line segment and the set of points above the old segment. It selects the set of points above its line segment and finds the extremal point with respect to that segment. This point is guaranteed to be on the convex and when added to the approximation initiates the next level of recursion. Each branch of the recursion terminates when its input is the empty set.

   These sub-problems generated by the recursion are solved in parallel. As above, selection and maximum are done in parallel if the size of the input set is large enough. If the largest sub-problem of each step in the recursion is a bounded fraction of the size of its parent problem then the total depth of the tree will be proportional to $\log H$ where H is the number of points on the hull. Since the expected number of hull points will be proportional to $\log N$ [PS86] the total expected time to execute the algorithm should be proportional to $\log\log N(N/P + \log P)$.

Note that the problem statement says that the points are distributed uniformly in a square and that there are only 1000 of them. By [PS86] this means that the expected number of points on the hull will be approximately twenty. Given the granularity of parallelism available on the Butterfly this is a very small problem instance and it is difficult to justify a parallel solution for it. We have therefore taken the licence to solve much larger problem instances and to look at other distributions of the points.

Although we have not attempted to tune the program to take advantage of the details of the problem statement, we are taking advantage of the square region by using the line $x = y$ to determine the direction in which to search for the initial extremal points.

Note also that our initial implementation does not use the above mentioned "tree of arrays" representation of a set. As a result there may be contention for adding points to the set. This contention may be contributing a linear time component to the running times. Once we have had the time to run the experiments needed to understand the current implementation better we can experiment with changing the representation of a set.

4 Evaluation of the Convex Hull Program.

It is difficult to evaluate the effectiveness of parallelism in geometry problems because the sequential Quickhull algorithm is so good. Craig McGowan provided the following set of single processor Quickhull timings (in seconds) obtained on several varieties of Sun workstation.

Points	2/50, 4 Meg	2/120 2 Meg	3/160C, 4 Meg
100	0.02	0.02	0.01
200	0.04	0.04	0.02
500	0.08	0.08	0.03
1000	0.16	0.16	0.05
2000	0.34	0.34	0.09
5000	0.84	0.84	0.24
10000	1.66	1.66	0.48
20000	3.34	3.34	0.98
50000	8.16	8.16	2.42
100000	16.30	16.30	4.87
200000	32.64	50.74	9.72
500000	182.95	308.51	37.23

These are Stuart Friedberg's comments on these experiments:

1. This Quickhull program is a CPU-bound task that carefully avoids 32-bit multiplies and floating point operations. The Sun-3's are roughly 4 times faster than Sun-2's. For CPU-bound tasks with int or long multiplies or with floating point, the Sun-3's should do even better. A simple program profiler for the Butterfly indicates that some programs spend more than 95 percent of their time in Chrysalis doing the software integer multiply necessary to compute array indicies. The 68020 processor, unlike the 68000 has a hardware 32-bit multiply. Thus it appears that a processor upgrade could have a significant impact upon execution speeds. The addition of a 68881 floating point coprocessor could have an even greater effect on speed in computations in which floating point and trigonometric functions are common.

2. The Sun 2/120's are Multibus-based, while 2/50's don't even have a bus. This makes IO comparisons hard between them and a Sun-3/160C, which is VMEbus-based. However, we can see that when both the 2/50 and 3/160C with the same amount of memory are thrashing, the Sun-3 still runs 6 times faster. It would be interesting to see a comparison between a /120 and a /160 with the same amount of memory and the same processor type.

Despite the excellent performance of the sequential algorithm the parallel version was able to use some parallelism effectively. Given our initial implementation using the sequential memory allocator, a Butterfly computes the convex hull of 10000 points in the following times:

Processors	Time	Speedup
1	7.60	1.00
2	4.31	1.76
3	3.22	2.35
4	2.54	2.99
5	2.06	3.68
6	1.81	4.18
7	1.73	4.38
8	1.54	4.91
9	1.48	5.13
10	1.42	5.32
11	1.20	6.32
12	1.24	6.11
13	1.37	5.52
14	1.17	6.44
15	1.20	6.33
16	1.15	6.60

These times (in seconds) reflect the actual computation time, excluding the time to load the program and the input data. As expected, the high overhead of managing the parallel implementation limits the amount of effective parallelism obtainable. Furthermore, the execution times do not decrease monotonically as processors are added. The source of this is likely to be some kind of scheduling or concurrency control artifact introducing serialization in a way that is very sensitive to the number of processors.

Note that a single Butterfly node executes the parallel implementation at about one sixth of the speed of a Sun 2 executing the straightforward sequential implementation. Part of the difference is due to hardware differences and part is due to overhead in accomodating potential parallelism. When 8 nodes are allocated to the problem the Butterfly outperforms the Sun 2, but at no point can it compete with the Sun 3. It is difficult to overcome the handicaps of lower single processor and memory speeds combined with the disadvantage of not having powerful parallel arithmetic in hardware.

To reduce the total amount of overhead and to eliminate a known significant source of "hidden" serialization a second version of the program was written that incorporated its own parallel memory management package. This second implementation performed as follows:

	Number of Points					
	1000		5000		10000	
Processors	Time	Speedup	Time	Speedup	Time	Speedup
1	1.08	0.99	4.29	1.0	8.01	0.99
2	.60	1.78	2.4	1.78	4.33	1.84
3	.46	2.30	1.8	2.38	3.27	2.44
4	.32	3.28	1.37	3.12	2.46	3.25
5	.29	3.70	1.16	3.68	2.00	3.99
6	.25	4.25	1.02	4.18	1.81	4.42
7	.26	4.09	.91	4.7	1.61	4.96
8	.24	4.36	.83	5.14	1.46	5.47
9	.22	4.87	.77	5.53	1.35	5.89
10	.22	4.75	.74	5.8	1.28	6.25
11	.20	5.20	.7	6.12	1.20	6.62
12	.20	5.38	.67	6.37	1.14	7.00
13	.19	5.48	.63	6.71	1.11	7.20
14	.20	5.31	.62	6.89	1.08	7.40
15	.19	5.46	.61	6.93	1.04	7.65
16	.19	5.42	.62	6.82	1.03	7.73
17	.18	5.83	.62	6.88	1.02	7.79
18	.19	5.54	.60	7.05	.99	8.06
19	.17	6.14	.6	7.05	.98	8.11
20	.17	6.00	.6	7.10	.97	8.19

The improved program is faster than the original. is able to use more processors effectively on average, and as processors are added the running time decreases.

5 The Voronoi Diagram.

Rather than compute the Voronoi diagram directly we would compute its dual. the Delaunay triangulation. We expect that a straightforward recursive parallelization can be performed upon the divide and conquer Delaunay triangulation program of Lee and Schacter [LS80]. We believe that this is the approach appropriate for the Butterfly.

The problem with this is that the final merge step will take time proportional to $\sqrt{N}$ on average. If all partitions are made so as to keep the merges proportional to boundary length, such as by alternating in the X and Y directions then the expected time could be $O(\log N \sqrt{N})$. The question is whether or not this can be improved on the Butterfly. The Aggarwal *et al* paper sketched a parallel merge step using $O(logN)$ time and $O(N)$ processors. Thus, it is clear that the merge can be sped up aymptotically on a PRAM. but at the moment it is not clear how to program it and how to design the data structures on the Butterfly so as to simulate the fine granularity implied by their paper.

The "divide" step of the algorithm requires partitioning the points into linearly separable subsets. In sequential implementations this is done by sorting the points along one of the coordinate axes. There is at the moment no general purpose sorting package for the Butterfly. Sorting is one of the most studied and fundamental computational problems, the fact that we still do not have a good, implemented solution on the Butterfly is indicative of the lack of maturity of the software environment on the machine.

6 The Euclidian Minimum Spanning Tree.

The EMST can be easily derived in sequential time proportional to N from the Voronoi diagram (Delaunay triangulation) [PS86] since the edges of the tree will be a subset of the edges of the triangulation.

Kwan and Ruzzo [KR84] survey "edge-adaptive" parallel algorithms for computing minimum spanning trees of arbitrary graphs. These have running times in the CREW PRAM model of

$O(E \log N/P$. Since the edges that need to be considered are a sub-set of the edges of the Delaunay triangulation the cost for the Euclidian minimum spanning tree once the triangulation is found will be $O(N \log N)$. As with the other two problems, this presupposes that we will be able to program efficiently shareable data structures representing dynamic graphs.

An alternative to using a PRAM style algorithm would be to use Bentley's [Ben80] optimal algorithm that uses $N/\log N$ processors in a fixed interconnection network. Since $\log 1000 \approx 10$ the a program that finds the EMST for 1000 nodes would potentially map very well onto a Butterfly of 100 nodes. In contrast to the PRAM algorithms mentioned above, Bentley's algorithm is designed for a set of simple processing elements that communicate over a fixed interconnection network. In particular, it is suitable for a VLSI implementation. While this avoids the problem of designing shareable dynamic data structures for graphs, the algorithm assumes a fine grained parallelism that depends upon very efficient inter-processor communication. As mentioned elsewhere in this collection of reports the SMP programming environment provides interprocessor communication in approximately two milliseconds. This is still too large in comparison to the amount of computation to be done at each node per message. The effect of communication overhead can be reduced by blocking several logical messages per physical message, but this increases the complexity of the programming effort. What seems to be needed here is some form of inter-processor *streams* interface.

References

[ACG*85] Alok Aggarwal, Bernard Chazelle, Leo Guibas, Colm O'Dunlaing, and Chee Yap. Parallel computational geometry (extended abstract). In *Proceedings 26th IEEE FOCS*, pages 468–477, Tucson AZ, October 1985.

[Ben80] Jon Louis Bentley. A parallel algorithm for constructing minimum spanning trees. *Journal of Algorithms*, 1:51–59, 1980.

[KR84] S.C. Kwan and W.L. Ruzzo. Adaptive parallel algorithms for finding minimum spanning trees. In *Proceedings of the 1984 International Conference on Parallel Processing*, pages 439–443. Bellaire, Mich., 1984.

[LS80] D.T. Lee and B.J. Schachter. Two algorithms for constructiong a delaunay triangulation. *Int. J. Comput. Inf. Sci.*, (3):219–242. 1980. Also appeared as GE Technical Report 79ASD007, July 1979.

[Mel84] Kurt Melhorn. *Data Structures and Algorithms. Volume 3: Multi-Dimensional Searching and Computational Geometry. EATCS Monographs on Theoretical Computer Science.* Springer–Verlag, New York, 1984.

[PS86] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry. An Introduction.* Springer–Verlag, New York, 1986.

[Sny86] Lawrence Snyder. *Type Architectures, Shared Memory, and the Corollary of Modest Potential.* Technical Report TR 86-03-04, Department of Computer Science. University of Washington, Seattle, WA, 1986. To appear in Annual Review of Computer Science. Vol. 1, 1986.

Chapter Seven: Visibility Calculations

Triangle Visibility

Christopher Brown, Liudvikas Bukys, Michael Scott

16 October 1986

1. The Problem

The problem as stated is ambiguous. We take it to mean "report visible *vertices*". The size of the problem was well-chosen, providing a reasonable exercise that exposed limitations in algorithms and data structures. The problem specifies floating point, but we use integers. The lack of hardware 32-bit arithmetic in the 68000 is enough to confuse the architectural issues, and the lack of floating point is such an obvious and important one that it should not be further folded into the problem. There is evidence that even the integer multiplication in array index calculations on the 68000 is inefficient enough to distort the architectural picture. Since there is an easy fix to this problem on the Butterfly, issues such as contention, the number of processes supported, and so forth are more interesting.

2. The Approach

A shared memory, SIMD-like, Uniform System virtual architecture fits with the algorithm we chose to implement, which is a quadratic all-against-all comparision of points against triangles for visibility. Below we discuss variations *on this theme*, and give the justification for the approach that we ultimately implemented. There is of course substantial room for more work on this algorithm, and there are other approaches as well.

3. Three Algorithms

We describe two algorithms, PointTri() and TriTri(), and a hybrid variant. PointTri() is basic.

```
PointTri(Points, Triangles)
{
for_each Point
        for_each Triangle
                if Occludes(Triangle, Point) mark Point "Hidden";
}
```

PointTri() can be enhanced in obvious ways to prune the full $3N^2$ search: In Occludes(), quit early and continue the loop as soon as it is determined that a triangle cannot hide a point. As soon as a point is found to be occluded, break the inner loop. Empirically, it seems this pruning gives a factor of two speedup (random inputs) over the full search. This speedup motivates TriTri(), which removes (a subset of) occluded triangles as well as occluded points from consideration, thus cutting down on the length of both inner and outer loops.

3.1. Point against Triangle

For PointTri(), computation falls into two stages, called 1 and 3 for consistency with TriTri().

(1) Stage 1 is a linear setup stage in which four planes are calculated for each triangle: the plane in which the triangle lies and the plane through the origin and each triangle side. These planes are kept together as a row in a Triangle array, and each point is a row in a Point array.

(2) Stage 3 is the quadratic (doubly-nested for_loop) comparison of points with triangles referred to above. Occluded points are marked "Hidden."

3.2. Triangle against Triangle

In TriTri(), Stage 1 has more to do, there is a Stage 2, and Stage 3 is more complicated. The idea is to sort triangles by order of their likelihood of obscuring other triangles, and to consider them in this order, getting the maximum pruning advantage. The right quantity to sort on is the amount of volume in the (1000 x 1000 x 1000) cube of space shadowed by a triangle (hidden by it from the origin). A quick approximation to this volume is quite good enough (details below).

(1) Stage 1 computes the triangle's approximate shadowed volume as well as its planes.

(2) Stage 2 sorts triangles by their approximate shadowed volume.

(3) Stage 3 calculates hidden points and a subset of hidden triangles: triangles and points each have a "Hidden" mark. Without solving the full hidden line problem, it is safe to mark a triangle "Hidden" if it is hidden by another single triangle. The control structure of the nested loops is slightly more complex because of the extra break condition (a triangle is hidden). The same Occluded(Point,Triangle) function is still the inner-loop workhorse.

3.3. Hybrid -- Point against Sorted Triangles

The idea here is add TriTri()'s Stage 2 to PointTri(), to sort triangles by shadowed volume, again hoping the extra work pays for itself with an increased pruning factor.

4. Some Geometric Details

Points are represented by three integers (x,y,z), planes by four integers (A,B,C,D) from the plane equation $Ax+By+Cz+D$. For Stage 1, if **u** and **v** are "triangle edge" vectors (the difference between two vertex points) then $\mathbf{u}\times\mathbf{v}$ is a vector (A,B,C), giving three plane coordinates. The fourth coordinate is given by $D=-(x,y,z)\cdot(A,B,C)$. A,B,C,D need not be scaled to make (A,B,C) a unit vector for the purposes of this work, and integer arithmetic is sufficient to hold all significant digits. Further, for the edge plane calculations the origin is a vertex, so **u** and **v** are just triangle vertices and D=0.

For Stage 2, the triple product $V = \mathbf{x} \times \mathbf{y} \cdot \mathbf{z}$ gives a volume proportional to that enclosed between the origin and the triangle. The strange quantity e, simply the sum of all the nine x, y, and z components of the three vertex points, is taken approximately to vary monotonically with the distance from the origin to the centroid of the triangle. $(V/e^3) - V$ is the final approximation of trucated shadowed volume, up to some scaling constants. The cost of the whole approximation is 17 multiplies and 14 adds.

This approximation was compared with a much more elaborate one that projects the triangle onto the unit sphere, computes the area of the resulting triangle, computes the centroid exactly, and then computes the shadowed volume fairly precisely truncated by a sphere of radius 1.42. PointTri was modified to do a non-pruned triangle-point computation and to report how many points were occluded by each triangle. This information was used to establish the "correct" order for the triangles -- increasing number of occluded points. The sort by both the shadowed-volume criteria was quite successful and yielded a (surprisingly) good approximation to the "correct" sort. The availability of a relatively cheap and effective sorting criterion paved the way for a fair experimental investigation of the sort's utility, which was easier than a responsible theoretical analysis.

For Stage 3, the central visibility calculation for point x and triangle (A,B,C,D) is $\mathbf{d} = \mathbf{x} \cdot (A,B,C) + D$. If the d for any of the four planes is negative (with my sign conventions) the point is on the unshadowed side of the plane. Thus in the worst (point hidden) case there are three multiplies, three adds and a comparison for one plane (with nonzero D) and three multiplies, two adds, and a comparision for each of three planes (with D zero). Any negative result terminates the calculation with a "Not Hidden By This Triangle" result.

5. Early Experiments

Uniprocessor implementations of the three algorithms established that the pruning accomplished by TriTri() and the Hybrid PointTri() was not worth the effort. Sorting was done by the UNIX qsort() utility. With TriTri() in the worst case, three times the number of points must be checked as in PointTri(), and the number of triangles that are wholly hidden by other single triangles is not very large. The Hybrid algorithm produced times comparable with PointTri(), but up to 1300 points no clear dominance was established, so it appears that sorting just pays for itself in the Hybrid PointTri(). Of course a fast parallel sort could change the results on the Butterfly. The linear Stage 1 (setting up the geometry) is, as expected, extremely fast compared to the quadratic Stage 3. The pruning provided by quitting early in the Stage 3 of PointTry() yields about a factor of two in speed.

6. Initial Uniform System Implementations

The algorithm PointTri() lends itself naturally to a Uniform System implementation. The Uniform System gives parallel for-loop capability. The implementation simply parallelized the main loops in Stages 1 and 3. The

resulting code came to 450 lines for Stage 1 and 185 lines for Stage 3. It was run in several versions on the three Butterfly Parallel Processors at the University of Rochester. Representative code appears in the last section.

Version 1 scattered the (point, visibility) and triangle arrays as usual. Version 2 locally cached the row pointers to these arrays. Version 3 locally stored the point coordinates and cached the row pointers to the triangle and visibility arrays.

7. Times

Comparative timing shows that the VAX 750 is approximately 10 times as fast on this job as a single node in our (not floating-point platform) Butterfly computer.

1000 Triangles VAX and Butterfly (Version 1) Times	
Configuration	Time in Seconds
1 VAX 11/750	97
1 Bfly Node	1035
2 Bfly Nodes	520
4 Bfly Nodes	261
8 Bfly Nodes	131
16 Bfly Nodes	67
32 Bfly Nodes	35
64 Bfly Nodes	25

1000 Triangles on Butterfly (8 Nodes) Effect of Caching (Versions 1, 2, 3)	
Caching Version	Time in Seconds
8 Nodes, Version 1	131
8 Nodes, Version 2 (row ptrs)	79
8 Nodes, Version 3 (Vers. 2 + points)	67

8. Further Uniform System Implementations

Two revised versions of the PointTri() algorithm were implemented by Bukys with improved results. Some of the improvements are due to the release of the new Butterfly compiler; others are due to some tuning of the implementation.

The major difference between this implementation and the previous ones is the memory-sharing strategy. Since the algorithm uses a brute-force $O(n^2)$ strategy, each point-processing step may access every triangle data structure. These computations will clearly run fastest when every processor has its own local copy of the data structures describing triangle geometry. Such sharing is possible because the data strctures are computed only once and can be treated as read-only and static thereafter. Unfortunately, it takes time to replicate the data structures.

This program illustrates the resulting tradeoff dramatically: Replicating read-only data takes some time, but makes the computation run fast; sharing some data reduces replication overhead but increases computation time due to remote references and (perhaps) memory contention.

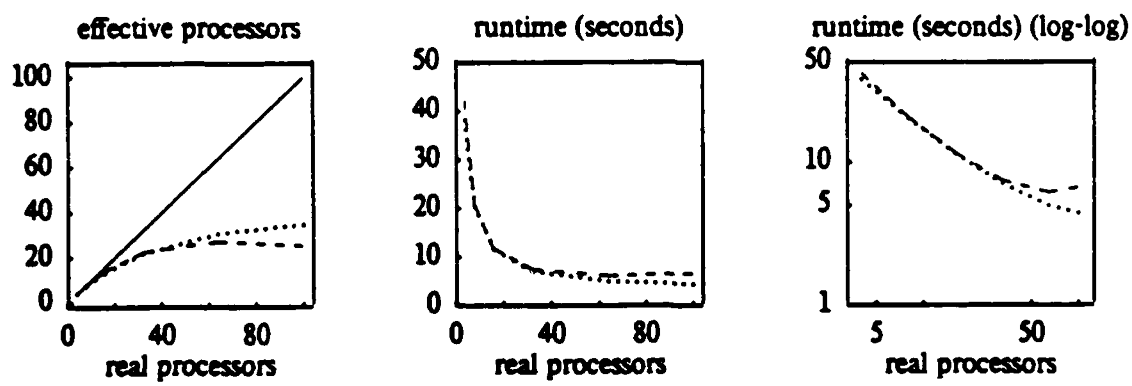
Further, the method of replication has a significant impact on runtime. The Uniform System implements two mechanisms for automatic replication: SharePtrAndBlk, which makes all copies from a single master, and ShareSM, which makes copies from previous copies, distributing the replication load among memories in a tree-like fashion with resulting logarithmic speedup. While the two procedures implement essentially the same function, their performance varies drastically. In the table below, compare the times in the rows "replicate triangle points" and "replicate planes" for the two implementations. Experiments have shown that the simple SharePtrAndBlk procedure works well for small pieces of data (under 2200 bytes), while the fancier ShareSM begins paying for itself for pieces of data larger than that. Unfortunately, the current Uniform System package provides the ShareSM procedure in a form suitable only for sharing row pointers of matrices. It would be a good idea to make both Share procedures use a data-size criterion for choosing replication method.

The following table breaks down the time spent in different phases of the computation for a 100-processor run of the algorithm. The final times were 6.5 seconds and 4.1 seconds, with the difference mainly accounted for by the different system calls implementing replication (shown in the "replicate planes" row. A constant 1.4 seconds is spent in generating the data (serially for replicability). The table illustrates that in the two computational steps (compute triangle parameters and detemine obscuration of points by triangles) typical speedups were almost linear (note the processor efficiencies of between 69% and 86% in the rows "make triangle and edge planes" and "visibility"), even with 100 processors running. However, the cost of replication is significant, and actually slows down the computation in the SharePtrAndBlk implementation for large numbers of processors. See the listing of times and graphs below. An obvious further tuning is to explore the tradeoff and find the amount of maximally efficient sharing.

Times for 100 Processors, 1000 Triangles

step	SharePtrAndBlk()		ShareSM()	
	effcy	*time(secs)*	*effcy*	*time(secs)*
initialize benchmark (100 procs)	-	4.186	-	4.200
allocate triangle pts	3.9%	.000	3.9%	.000
make 1000 random triangles	3.9%	1.473	4.0%	1.463
replicate triangle points	.1%	.669	1.1%	.124
allocate planes, replicate ptrs	1.1%	.026	1.0%	.030
make triangle & edge planes	74.1%	.035	68.7%	.033
replicate planes	.1%	2.368	.6%	.590
visibility: 256 points visible	86.4%	1.839	77.0%	1.852
FreeAll	2.3%	.048	.6%	.062
TOTAL (w/o initialization)	25.2%	6.458	34.6%	4.154

Speedup Graphs for Triangle Visibility



The graphs above were produced from the following raw data.

Raw Data for SharePtrAndBlk version:

[4] time = 676665 ticks = 42.29 sec; ep = 3.9; eff = 0.9999
[8] time = 332146 ticks = 20.75 sec; ep = 8.1; eff = 1.0186
[16] time = 188039 ticks = 11.75 sec; ep = 14.3; eff = .8996
[32] time = 120625 ticks = 7.53 sec; ep = 22.4; eff = .7012
[64] time = 99205 ticks = 6.20 sec; ep = 27.2; eff = .4263
[100] time = 107235 ticks = 6.70 sec; ep = 25.2; eff = .2524

Raw Data for ShareSM version:

[4] time = 610096 ticks = 38.13 sec; ep = 3.9; eff = 0.9999
[8] time = 324462 ticks = 20.27 sec; ep = 7.5; eff = .9401
[16] time = 184055 ticks = 11.50 sec; ep = 13.2; eff = .8286
[32] time = 113449 ticks = 7.09 sec; ep = 21.5; eff = .6722
[64] time = 79820 ticks = 4.98 sec; ep = 30.5; eff = .4777
[100] time = 70453 ticks = 4.40 sec; ep = 34.6; eff = .3463

9. A Pipeline Algorithm in LYNX

A systolic approach to solving the triangles problem was suggested by Peter Dibble and refined and implemented by Michael Scott. Triangles are fed into one end of a process pipeline. When they emerge at the other end, their vertices are marked "visible" or "hidden." In the simplest version of the algorithm, there are an equal number of triangles and processes. A special process feeds the head of the pipeline with triangles whose vertices are all marked "visible." An additional, marker triangle is fed through last. Before the pipeline begins operation, a preliminary phase of the algorithm precomputes, in parallel, the coefficients of plane equations that will be needed to determine if a point is obscured.

Each pipeline process keeps the first triangle that reaches it. It passes subsequent triangles on to its successor, marking as hidden any previously-visible vertices that are obscured by the original triangle it kept. When the marker triangle arrives, the process passes its personal triangle on to its successor, followed by the marker. Triangles emerging from the end of the pipeline have been compared against every other triangle.

An optimized version of the algorithm attempts to hasten comparisons that are most likely to find obscured points. In addition to computing plane equations, the initialization phase also computes the approximate volume of space shaded by each triangle. Each pipeline process compares the shaded volume of each newly-received triangle against the shaded volume of its personal triangle. If the new triangle is "bigger," it swaps them, keeping the new triangle and passing the old on to its successor.

The optimization is particularly important in practice, as there are many fewer processors than triangles. If each of the early stages of the pipeline is run on a different processor, and if each of the triangles in those early stages shadows a large volume of space, then the odds are good that relatively few obscuration tests will be needed in later stages of the pipeline.

Scott coded the pipeline algorithm in LYNX, a message-based language available on the Butterfly here at Rochester. The original version, with one process per triangle, does not accommodate large problem instances, because of operating-system limitations on the number of processes per processor. Scott then made a second implementation in which the pipeline, having exhausted processors, doubles back *through existing processes* as many times as necessary to provide one stage per triangle. If there are *K* processors, then processor *I* contains stages *I*, *2K-I+1*, *2K+I*, *4K-I+1*,

The multiple-stages-per-process implementation is significantly more complicated that the original version. It has uncovered a bug in the Chrysalis operating system which, in the limited time available to us, we have not yet been able to correct. For 200 triangles (the largest problem size that does not trigger the Chrysalis bug), the algorithm completes in about 15 seconds with a 100-processor pipeline.

10. Architectural Implications

Floating point processing (and hardware integer processing) is necessary. BBN currently provides an upgrade package (M68020-68881 daughter board) that we hope to acquire.

The Butterfly can present many abstract architectures to the user. For the Uniform System algorithm, a high-level and fairly superficial list of observations follows. In the US, memory allocation causes dramatic serialization: Parallel allocation would help. Carla Ellis and Tom Olson at Rochester are studying that problem. A geometric coprocessor or preprocessor for fast computation of trigonometric, roots, vector and matrix operations would be useful (the WARP comes to mind here). Better debugging support, from symbolic tools down to certain microcode enhancements would speed the development cycle. A combining switch would reduce memory contention, which may be a bottleneck in this computation.

The ability to share (copy) data quickly between processors would make a significant difference in this implementation, since in the final versions much data copying was done to increase the locality of computations on individual nodes. There is clearly a tradeoff in the current architecture between memory contention and the cost of massive data copying.

Serialization within the system is costly. Some of it can be avoided by the clever user, but some of it (such as memory allocation) should be parallelized by the system.

In the LYNX algorithm, we believe that the inefficiency of the pipeline is due primarily to the relative expense of sending a message (on the order of 2 or 3 milliseconds) compared to the amount of real work performed per message. To amortize the cost of message-passing, we would need in a practical implementation to pass more than one triangle in each message. Like the need to package more than one pipeline stage in a process, the need to package more than one triangle in a message complicates the implementation considerably, and suggests that the parallelism of the pipeline is too fine-grained for effective expression in LYNX. Unfortunately, there is no software package currently available on the Butterfly that supports orders of magnitude more processes than processors. The Uniform System does not count in this regard because its tasks cannot be suspended and are therefore able to synchronize only by busy-waiting.

We have begun to realize that a large and important class of problems can be solved by devoting a process to each of a very large number of objects. Many parallel algorithms in the current literature are of this flavor: the geometric problems in the benchmark provide more examples. To aid in mapping algorithms from the literature onto the Butterfly, a language akin to Mesa or Concurrent Euclid would be a very useful tool. Ada would also work, though probably not as well.

11. Stage 3, Version 3 Of PointTri() and the Uniform System

Besides making the inner loop of the computation explicit, this code segment illustrates several points. First, it shows that the Uniform System is easy to use: Both the US and SMP libraries give the new user very rapid startup. Second, it reveals that the Butterfly architecture is not actually a shared memory machine. There are several standard practices to reduce memory contention, the most common being local caching or copying of data. These practices acknowledge local memory. Below, local copies are made in the initializing routines GenericInit() and TInit(), and in the inner loop routine TriHidesPt(). Also the point array ShrPts[] has been copied to every node. Further, US has some hidden serializations: the storage allocator works sequentially, using global locks. The Allocate() call in ForAllTriangles() is natural but can (and should) be eliminated. Implicit in this example is that the ease of Butterfly programming and the flexibility of the architecture place a burden on the designer to come up with an efficient algorithm and data structures -- the architecture does not dictate them.

```
/*************STAGE 1 NOT SHOWN HERE ********************************/
/*************STAGE 3 -- CHECK  POINTS AGAINST TRIANGLES************/

CheckPoints() /*outer parallel for loop -- for all points */
     {
     GenOnIndex(GenericInit, ForAllTriangles, p, 3*(p->N));
     }

ForAllTriangles(Arg, PointNdx) /* inner loop -- for all triangles */
```

```
        Problem *Arg;
        int PointNdx;
        {
        Problem *t;
        int i;

        t = (Problem *) Allocate(sizeof(Problem));
         /*this Allocate should be avoided: allocation is done serially */

        t-> N = myproblem.N;
        t->ThisPointNdx = PointNdx;
        t-> Vis = myproblem.Vis;
        t-> Tris = myproblem.Tris;
        t->ThisVis = -1;              /*create problem structure */

        GenOnIndex(TInit, TriHidesPt, t, t->N);  /*parallel for loop */
        }

TInit(Arg)    /* Standard practice. make local copies of
              global scattered data to avoid contention. In this case
              copy row pointers and problem structure */
Problem *Arg;
{
static int *vis[POINTS];
static int *tris[TRIANGLES];
block_copy(Arg, &MyTProb, sizeof(Problem));
block_copy(MyTProb.Tris, tris,(MyTProb.N)*sizeof( int *));
block_copy(MyTProb.Vis, vis,(MyTProb.N)*3*sizeof( int *));
MyTProb.Tris = tris;
MyTProb.Vis = vis;
}


TriHidesPt(Arg, TriNdx)  /* inner loop computation: does triangle hide pt? */
        Problem *Arg;
        int TriNdx;

        {
        int offset, MyX, MyY, MyZ, PlaneNdx;

        if( MyTProb.ThisVis == 0) return; /*is point already invisible? */

        offset = (MyTProb.ThisPointNdx)*POINTCOLS;
        MyX = ShrPts[offset];
        MyY = ShrPts[offset+Y];
```

```
	MyZ = ShrPts[offset+Z]; /*get point x, y, z */


	block_copy(MyTProb.Tris[TriNdx],MyTriangle,(TRICOLS)*sizeof(int));
		/* make local copy of scattered data */

	if( (MyX * Coord(TRIPLANE,A) /* dotproduct with plane of triangle */
		+ MyY * Coord(TRIPLANE,B)
		+ MyZ * Coord(TRIPLANE,C)
		+ Coord(TRIPLANE,D)) <= 0)
		return;		/*not hidden -- quit */

	for (PlaneNdx = EDPLANE1; PlaneNdx <= EDPLANE3; PlaneNdx++)
		/*dot with 3 planes of edges */
		{
		if( (MyX * Coord(PlaneNdx,A)
		+ MyY * Coord(PlaneNdx,B)
		+ MyZ * Coord(PlaneNdx,C)
		<= 0) )
		return;			/*quit early if not hidden*/
		}
	/*point hidden if get to here*/
	MyTProb.ThisVis = 0;
	MyTProb.Vis[MyTProb.ThisPointNdx][0] = 0;
		/*set local and global visibility */

	} /* end TriHidesPts */

/********************MAIN ROUTINES ****************/

FullJob()
	{
	MakeTriangles(); /*do Stage 1 */
	CheckPoints();  /* do Stage 3 -- see above */
	FreeAll();	/* clean up */
	}

main()
	{
	InitializeUs(); /*Uniform System Initialize*/
	MakeShrPts();		/* generate random 3-D triangle vertices */
	TimeTest(SetUp, FullJob, MyTestPrint);
	 /*run algorithm, get times on different numbers of processors */
	}
```

Chapter Eight: Graph Matching

see: Butterfly Project Report 14

Chapter Nine: Minimum-Cost Path

Minimum-Cost Path

Brian D. Marsh and Thomas J. LeBlanc
August 1986

1. Introduction

We describe an implementation of the *minimum-cost path problem* on the BBN Butterfly using the SMP message-passing library developed at the University of Rochester. The problem statement for finding the minimum-cost path is as follows:

> *The input is a graph G having 1000 vertices, each joined by an edge to 100 other vertices selected at random, and where each edge has a nonnegative real-valued weight in some bounded range. Given two vertices P,Q of G, the problem is to find a path from P to Q along which the sum of the weights is minimum. (Dynamic programming may be used, if desired.)*

Given this problem statement, it is ambiguous as to whether we are required to solve the *all-pairs-shortest-path problem*, which then allows the user to query the result regarding individual pairs of nodes, or whether we are to solve the *single-source-shortest-path problem* for a particular pair of nodes. Given that dynamic programming was specifically mentioned in the problem statement and is normally used to solve the *all-pairs-shortest-path problem*, we felt constrained to implement that problem, despite the fact that we believe the *single-source-shortest-path problem* has a more interesting parallel solution and would better exhibit the flexibility of the BBN Butterfly architecture. In the following sections we describe our parallelization of Floyd's algorithm for the *all-pairs-shortest-path problem*, an implementation of the algorithm on the Butterfly using the SMP message-passing library package, and our performance results.

2. A Parallelization of Floyd's Algorithm

We chose to implement a parallel version of Floyd's dynamic programming solution to the *all-pairs-shortest-path problem* [1]. The input graph is represented by an adjacency matrix. An entry, [i,j], corresponds to the weight of the edge from vertex i to vertex j. Nonexistent edges are denoted by a symbol representing infinite distance.

During execution each entry of the matrix corresponds to the cost of the minimum-cost path between two vertices. Initially, only those vertices that share an edge have a path of finite cost. Floyd's algorithm iterates over each row of the matrix, finding successively lower cost paths. During the k'th iteration, the algorithm computes the cost of the minimum-cost path between all pairs of nodes, i and j, that pass through no vertex numbered greater than k. For a graph with N vertices, N iterations are necessary. Therefore, the algorithm is $O(N^3)$. The code for the algorithm is as follows:

```
for k := 1 to N do
  for i := 1 to N do
    for j := 1 to N do
      if A[i, k] + A[k, j] < A[i, j] then
        A[i, j] := A[i, k] + A[k, j]
      end if
    end for
  end for
end for
```

An obvious parallelization of this algorithm results from treating each **for** loop as a parallel **for** loop. However, the granularity of the innermost loop is not large enough to justify the overhead of process allocation in the Butterfly. For this reason we chose to use the processing of an entire row as the unit of granularity for parallelism. We divided the problem matrix uniformly among the available processors, so that each processor has some subset of rows in the matrix. Since the size of the input graph is defined to be on the order of 1000 vertices, each processor must iterate over approximately 10 rows. The code for each process is:

```
for k := 1 to N do
  if row k is local then
    broadcast row k
  else
    receive row k
  end if
  for each local row i do
    for j := 1 to N do
      if A[i, k] + A[k, j] < A[i, j] then
        A[i, j] := A[i, k] + A[k, j]
      end if
    end for
  end for
end for
```

The primary data dependency in this algorithm is that all processes need a specific row at the same time, a row whose values are dependent on past computation. This synchronization constraint forces the processes in the algorithm to run in lockstep. On the k'th iteration, each process computes the optimal paths for its local rows using the values stored in row k. Computation cannot proceed until these values are known. The implementation, therefore, must have an efficient broadcast mechanism. For this reason, among others, we chose to implement the algorithm using the SMP library package.

3. An SMP Implementation of Floyd's Algorithm

An implementation of the *all-pairs-shortest-path problem* was done in C using the SMP library package developed at the University of Rochester [3]. SMP is a message-based programming environment for the Butterfly. Processes are dynamically created within SMP *families*. Interprocess communication within a family is based on asynchronous message-passing (send/receive) according to a fixed communication topology. When using SMP the programmer sees a small set of procedure calls for creating processes, specifying interconnection topologies, and sending messages. The details of the Chrysalis operating system needed to implement processes and communication are hidden. The programmer is free to concentrate on the issues pertaining to the application, rather than the underlying primitives.

There were several reasons for choosing SMP for this application. The most important reason is that our experience with a similar application [4] had shown that exploiting data locality could lead to significant performance advantages when compared with the shared memory approach of the Uniform System [2]. That is, storing a subset of the rows in memory local to the process that will modify those rows and exchanging rows in messages requires less communication than storing the rows in a globally shared memory. Another reason for using SMP is that broadcast communication, which is used in our algorithm, is directly supported. Finally, we were able to use this application to gain additional experience with SMP.

Our parallel version of Floyd's algorithm does not make full use of the tree of dynamic process structures available in SMP. In our implementation, a single parent process is responsible for creating a child process on each processor. Each child process is given some subset of the rows in the initial adjacency matrix. On the k'th iteration, each child process receives a message containing row k and computes new values for its local rows. The process containing row $k+1$ then broadcasts that row to all its siblings to start the next iteration.

The *send* primitive of SMP accepts a list of destination processes, therefore, both broadcast and multicast can be done directly in SMP. The SMP implementation of *send* is such that the cost of sending to one sibling (or to the parent) is the same as sending to 100 siblings. In each case, the message is copied to a buffer local to the sending process and flags are set indicating the intended recipients. Using the SMP *receive* primitive, destination processes can inspect the shared buffer to determine if there is a message directed to them. If so, the message is copied into the local memory of the receiving process.

One of the problems with broadcasting in SMP is that the Butterfly provides no hardware support for simultaneous communication with multiple destinations. In SMP each potential recipient of a message must map the message buffer into its local address

space to check for a message. Since each process in our algorithm is expecting to receive rows from every other process, the source list of each *receive* operation is very long. All the processes listed in the source list will have their message buffers mapped into the local address space during each iteration. This turns out to be extremely time consuming when the list is very long and, in an early implementation of our algorithm, was a dominating factor. Fortunately, we were able to exploit the inherent synchronization in our algorithm to reduce the overhead of broadcasting by minimizing the number of buffers examined on each iteration.

On each iteration, every process expects to receive a particular row. Despite the fact that rows are broadcast, the source for each row is known. Hence, in our implementation, we invoke the *receive* operation on the k'th iteration with a source list of size 1, namely, the process containing row k. This way, only one message buffer is mapped into the local address space on each iteration. We were able to improve performance by 50% using this approach. The performance of the resulting implementation is summarized in the next section.

4. Performance Results

The program to solve the *all-pairs-shortest-path problem* was developed on a host Vax 11/750 and downloaded to the Butterfly for execution. A sequential version was also implemented on a SUN workstation for comparison purposes. Coding and debugging the application program required about one week of effort by a graduate student; some additional time was spent debugging the SMP library.

For the purposes of the benchmark experiments, random graphs of various sizes were generated. We performed detailed experiments using two graphs: G1, a random graph containing 100 vertices with 10 edges per vertex, and G2, a random graph containing 300 vertices with 30 edges per vertex. We did not perform any experiments with the graph size given in the problem statement, 1000 vertices with 100 edges per vertex, for two reasons:

a) In order to demonstrate how well our implementation scales to multiple processors, we needed to run the algorithm with a varying number of processors and compare it to the single processor case. G2 requires 33 minutes of execution time on a single processor. By running significantly larger problems, we would be consuming a limited resource (Butterfly availability) and learn very little in return.

b) The cost matrix for a graph with 1000 vertices requires 4MB. While our Butterfly does have 1MB on each node, Chrysalis does not have good tools for creating and manipulating large objects that span multiple processors. The extra programming effort necessary to run such a large problem was not warranted.

In each of our test runs, only 100 processors were used, even though 120 processors were available. We did this so that all of our graphs would be uniformly distributed among the available processors. In this way, we eliminated the "tail end" effects that might otherwise distort our measurements.

Our performance results for finding the *all-pairs-shortest-path* solution for G1 and G2 on the Butterfly are shown in Figures 1-4. We have not included the initialization overhead in the results; only actual computation time was measured. The parent process in the SMP family was responsible for maintaining the timing results. All children synchronize with the parent, the clock is initialized, and all processes then begin computing. The results show the elapsed time between clock initialization and the final response from child processes.

These same graphs were also run on a SUN 2/50 workstation with 4MB of memory and a Vax 11/750 with 2MB of memory. G1 took 44.5 seconds on the SUN, 158 seconds on the Vax, and 69 seconds on a single Butterfly processor. G2 took 1205 seconds on the SUN, 2787 seconds on the Vax, and 1907 seconds on a single Butterfly processor. As can be seen in Figure 1, a small graph of 100 vertices can efficiently use 25 processors on the Butterfly (19 *effective processors*); additional processors do not provide much improvement in performance. The larger graph, G2, can make use of all 100 processors. In either case, only 2 Butterfly nodes are needed to significantly improve upon the sequential version on both the SUN and Vax.

5. Conclusions

To summarize the results of our experience with the *all-pairs-shortest-path problem*: a parallel version of Floyd's algorithm was easily implemented using SMP on the Butterfly and the resulting performance demonstrated nearly linear speedup using up to 100 processors. What follows are some comments about the choice of algorithm, software, and architecture.

The dynamic programming approach to the *all-pairs-shortest-path problem* is ideally suited to a vector machine: the Butterfly Parallel Processor has no special capability in this regard. Nevertheless, we felt this would be the easiest solution to implement in the limited time available. The fact that we were able to implement a solution to this problem on the Butterfly in a short period of time, a solution that demonstrated nearly linear speedup over the sequential version for large graphs, gives some measure of the flexibility of the Butterfly architecture. It would have been interesting to compare our experiences on this problem with similar experiences on the *single-source-shortest-path problem*, a similar problem with a more interesting parallel solution. Time did not permit this comparison.

Our experiences with the SMP system were very positive. A new graduate student was able to implement Floyd's algorithm in about one week of effort. The SMP library

dramatically reduces the learning curve for the Butterfly. However, the SMP library was only recently released, and we did encounter a few system bugs. All of the bugs were repaired in the same week. This effort did point out the need for some optimizations when handling source and destination lists in an SMP broadcast. We expect this will lead to slight modifications in the way SMP treats such lists. We also plan to add some additional routines that help in performing timing tests.

Our biggest problems with the Butterfly architecture continue to be related to memory management, in particular, the lack of segment attribute registers (SARs). SAR management was the source of most of the SMP bugs and is also the main difficulty in manipulating large objects. However, as we have gained more experience with the Butterfly, we have accumulated tools and techniques for solving most of the problems associated with SAR management. (For example, SMP incorporates a SAR cache for message buffers.) We expect that continued experimentation will yield additional solutions.

References

1. A. Aho, J. E. Hopcroft and J. D. Ullman, *Data Structures and Algorithms*, Addison-Wesley Publishing Company, 1983.

2. BBN Laboratories, The Uniform System Approach To Programming the Butterfly Parallel Processor, Version 1, Oct 1985.

3. T. J. LeBlanc, N. M. Gafter and T. Ohkami, SMP: A Message-Based Programming Environment for the BBN Butterfly, Butterfly Project Report 8, Computer Science Department, University of Rochester, July 1986.

4. T. J. LeBlanc, Shared Memory Versus Message-Passing in a Tightly-Coupled Multiprocessor: A Case Study, *Proceedings of 1986 International Conference on Parallel Processing*, (to appear) August 1986.

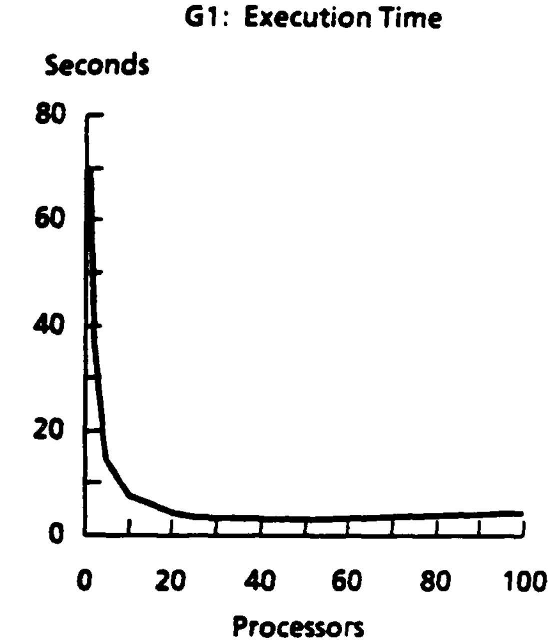


Figure 1

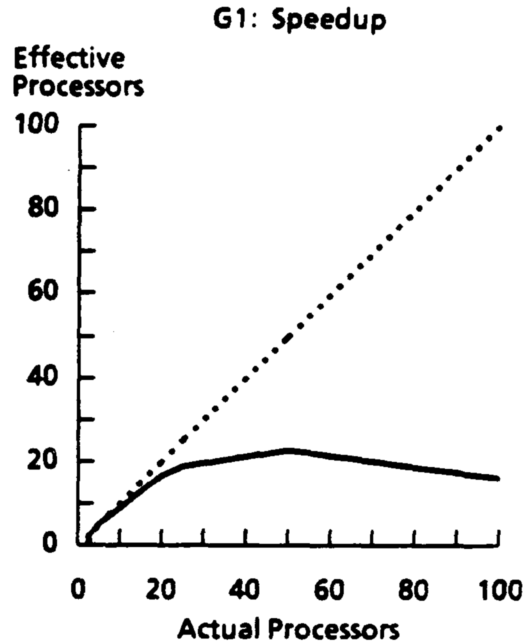


Figure 2

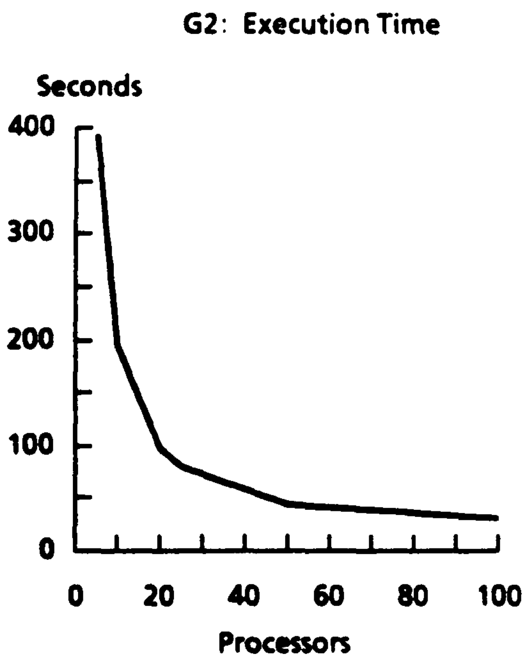


Figure 3

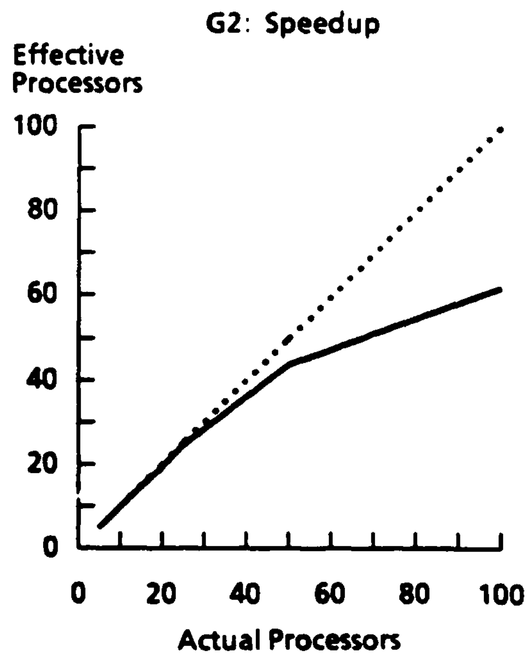


Figure 4

Appendix B-2

Rover Programmer's Guide

David J. Coombs*
coombs%cs.rochester.edu@relay.cs.net

The University of Rochester
Computer Science Department
Rochester, New York 14627

Technical Report DRAFT

May 1987

Abstract

This is the programmer's guide to Rover, a prototype active vision system [Coombs and Marsh 1986]. The system was built as a project for CSC 400 and CSC 446, but it is hoped the system will be used by others as a tool for investigating active vision problems in the laboratory. This guide describes not only the conceptual organization of the system, but also the existing source code.

The University of Rochester Computer Science Department supported this work.

***Thanks are due to Chris Brown, Brian Madden and Brian Marsh for their support and critiques of this document.**

Contents

List of Figures

1 Overview of Rover

Rover is the result of an attempt to build an extendible active vision system that is flexible enough to support vastly differing control strategies. Both top-down and bottom-up processing can be implemented in the Rover paradigm.

The current implementation of Rover performs the task of maintaining correspondences of distinctly colored spheres over time in a dynamically changing scene. Figure 1 diagrams Rover's main functional units and their relations to one another. Briefly, the Executive begins the work on each image frame by enqueueing a batch of "raster scans" in a static search pattern and then watches a clock to avoid spending too much time on any single image frame. (Other strategies could be employed to search the image frame for potential objects—see Section A.2.) The Raster Scan Cluster seeks light-colored "blobs" in the image frame that may be objects in the scene. The Object Discrimination Cluster scrutinizes the blobs pointed out by the R.S. Cluster and updates the world database. Thus the Executive performs limited top-down direction of computation, but otherwise, computation proceeds in a bottom-up fashion.

1.1 Inter-module Communication

A *module* is an open-loop[1] process that is invoked by another module. Modules that perform functions related to a particular goal are grouped together in a *cluster*. Rover maintains a priority queue of tasks waiting to execute. Arguments are enqueued with each module when it is entered on the queue of waiting tasks. Each module returns a special defined type as its result.

In the current implementation, module invocation (beyond the initial enqueuing of enough raster scans for the whole image frame) is driven by the results of each stage of processing. Each module performs its assigned task and enqueues the module whose work should be done next, based on the results of the current module. Information is passed between modules either by placing it in a global data structure (as in the raster scan cluster) or by wrapping it up and handing it to the next module as its argument (as in the object discrimination cluster).

A module may be composed of several functions, although it is crucial that each module execute and return quickly so the entire system is not bogged down by a sluggish module. (Rover is intended to be robust enough to adapt to a more rapidly changing environment by processing each frame less completely, and a module that runs a long time can cause the executive to lose track of the environment.) A module that needs to perform some auxiliary task to continue its computation is thus split into

1. a module to perform the initial computation,
2. a module to accomplish the auxiliary work,
3. and a module to be enqueued by 2 to conclude the work.

[1]Here *open-loop* means that the process can be dispatched without requiring the invoking module to monitor its progress.

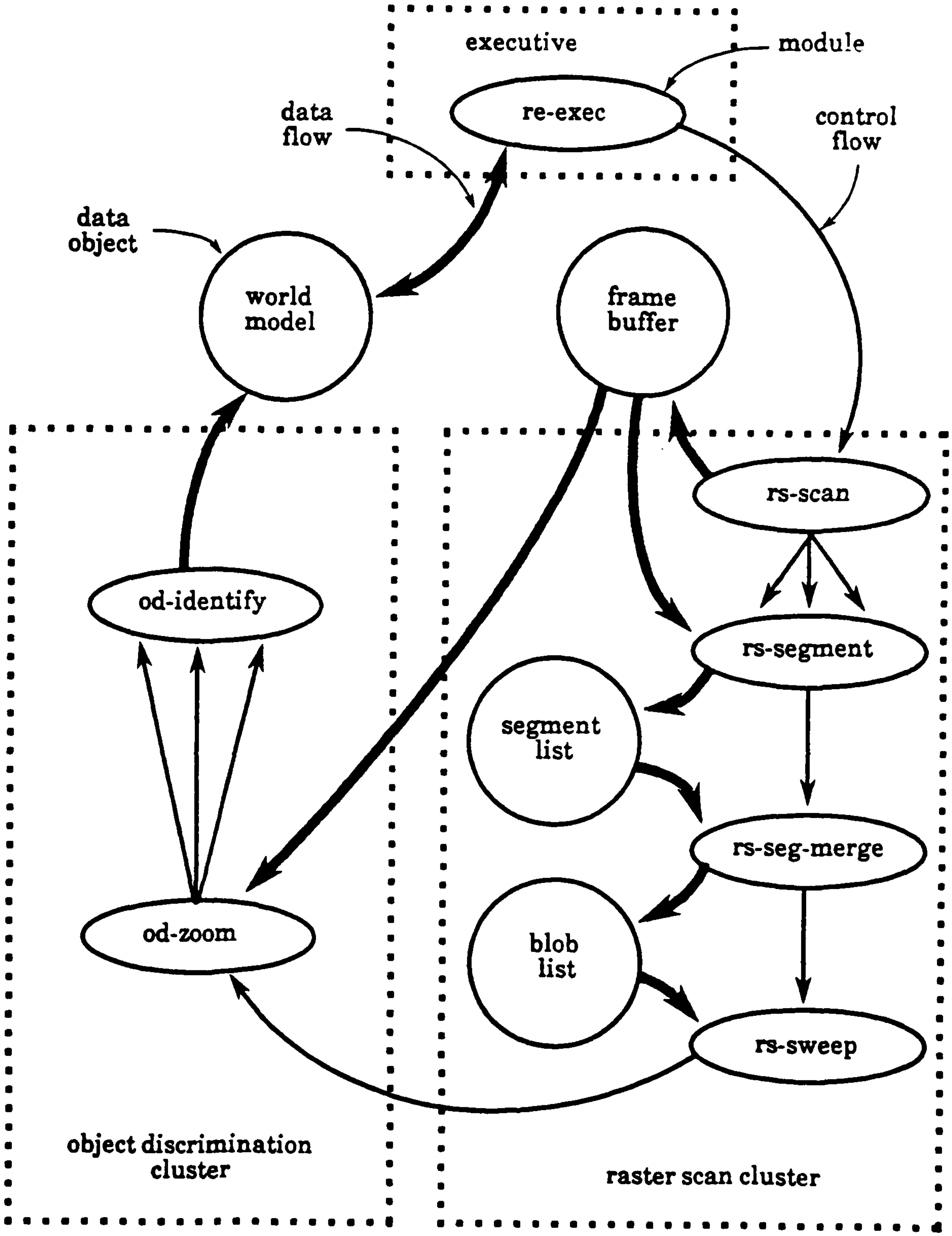


Figure 1: Functional Overview of Rover

A good rule of thumb is to design each module to perform a fairly straight-line function and then at a major decision point or functional transition to enqueue the next appropriate function to carry on.[2]

2 Existing System

Figure 2 depicts the topology of Rover's source code files. This should serve as a guide to the source code.[3] The clusters and libraries are briefly described in this section to provide an introduction to the code in the current implementation.

2.1 Executive (re_exec)

Rover begins execution in the Executive, setting the command line options and initializing the global data structures. The Executive then starts the first frame interval.

At the beginning of work on each frame, the Executive notes the time and enqueues enough raster scans to search the entire frame image at a predefined density (every 16 horizontal lines). When all interesting work has completed or time runs out on the current frame (the time limit is a compile-time system parameter) the Executive flushes the queue and degrades the confidence[4] of each object in the world database (DB). The confidence of an object is degraded the most if it was not updated at all during the previous frame interval. The confidence is degraded a little if its position and motion were updated based only on location and size correspondence with the image, and it not degraded at all if the "color" of the item was used in the correspondence check (*i.e.*the object's identity was "completely" verified, assuming each object has a unique "color").

2.2 Clusters

Raster Scan Cluster (rs_*)

This cluster scans the image frame coarsely to quickly identify blobs in the image and estimates the horizontal motion of each blob found. (Recall that the objects in the scene are assumed to move in horizontal planes.)

Rs_scan is called by the Executive to enqueue the scans initially; it also takes a new image in the frame buffer. (Actually, each image frame consists of a pair of images at one-half horizontal resolution. Rs_segment uses these two images to estimate the horizontal velocity of each detected segment. Subsequent modules use only the second image of the pair.)

[2] Although the modules in each cluster are currently structured as a progression of computational stages, Rover's facilities can support other organizations (*e.g.*a hierarchical system in which each cluster also has a queue of tasks and one module in each cluster acts as the "executive" for that cluster). In fact, Rover was intended as a vehicle for exploring various organizations for active vision systems.

[3] A listing of the code is available in TR ??? (are we sure we want to do this?).

[4] The *confidence* of an object reflects the quality of the data about the object.

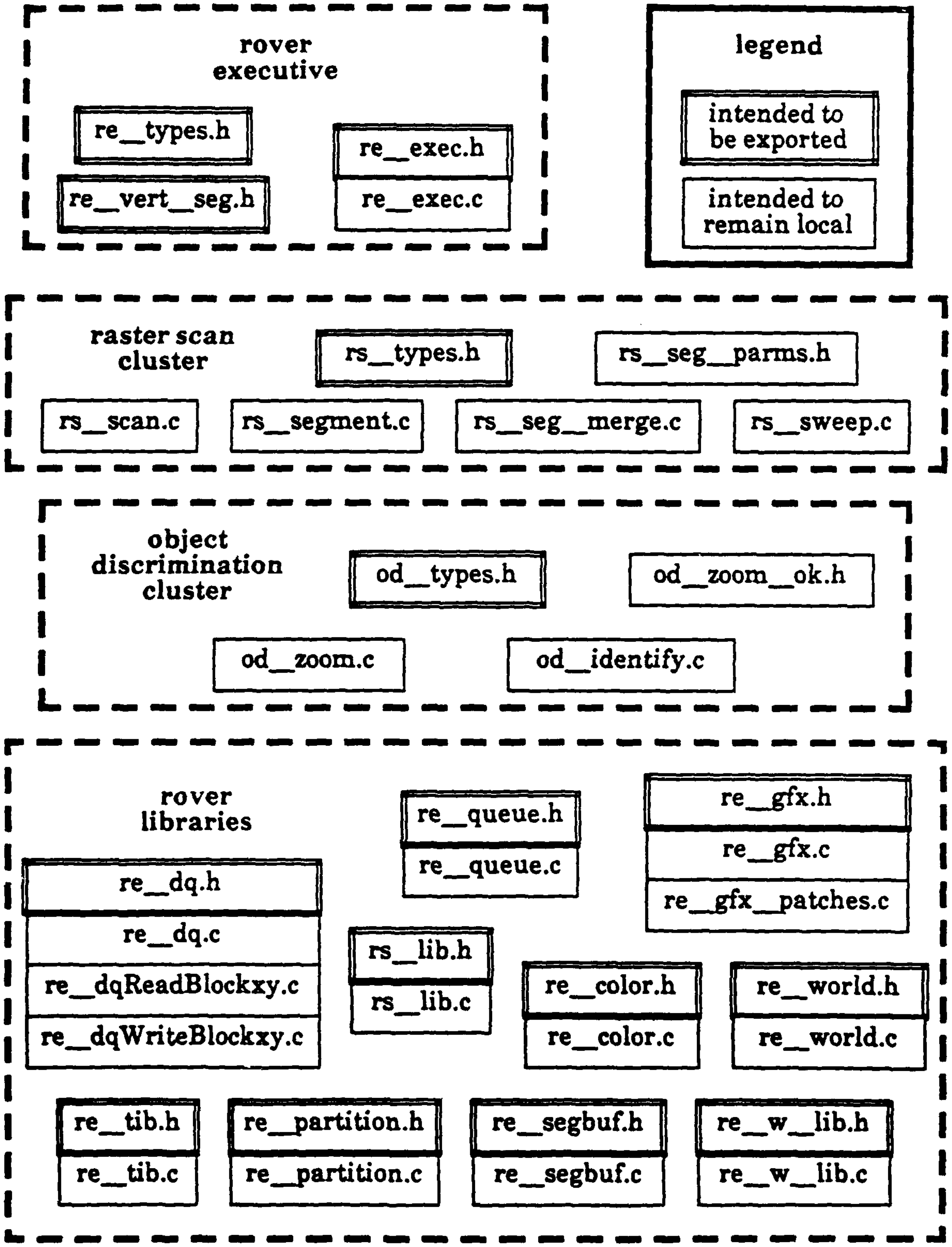


Figure 2: Rover's Source Code Files

Rs_segment recognizes a segment[5] in the pair of images if the segment appears in both frames and its images overlap spatially (if the two images are overlaid on one another). Each detected segment's horizontal velocity is also estimated, and all the segments detected on this horizontal raster line are recorded in the segment list. Then an rs_seg_merge is enqueued to work on these results.

Rs_seg_merge tries to merge each segment found on the indicated raster line either into an existing blob[6] or with another unmerged segment to create a new blob. A segment may merge into a blob or with another segment if they overlap and their estimated velocities are "close" to one another (as determined by a pre-defined percentage-error threshold). Then an instance of rs_sweep is enqueued.

Rs_sweep sweeps through the blob list to cap off existing blobs by noting a lack of segments in the expected positions on the rasters above and below the known extent of each blob. For each blob that is apparently bounded by background, an od_zoom is enqueued to examine that region of the image.

Object Discrimination Cluster (od_*)

This cluster examines at finer resolution each subimage identified by the Raster Scan Cluster as a potential object and uses its results to update the world database.

Od_zoom copies the indicated subimage from the frame buffer into a temporary image buffer. If an object appears to be "clipped" by any edge of the subimage (*i.e.*the object is not completely contained in the subimage) the subimage is discarded; otherwise, an od_identify is enqueued for each object found in the image.

Od_identify spatially back-projects its object-image onto the world DB to find objects already known in the world to which this image might correspond. If no objects of about the same size are found in the expected locations, or the confidence of the closest match is low (*i.e.*the accuracy of the information on this object is suspect) the image's "color" (mean and variance of brightness) is calculated and the world DB updated by the best match (if any are "close") in color, size, and *etc.*. If no existing object matches the image well enough, a new object is placed in the world DB.

2.3 Libraries

Task Queue Manager (re_queue)

This library provides the operations *new_queue, enqueue, dequeue_highest, dequeue, and q_flush* operations on instances of priority queues of tasks. A Module is enqueued as a pointer to a function, with a pointer to a structure that holds the function's arguments, a function to free the argument in case of a q_flush, and the module's priority.

[5] A *segment* is a bright section of a horizontal line that is surrounded dark sections.

[6] A *blob* is a set of vertically adjacent and horizontally overlapping segments whose velocity estimates are compatible.

Graphics Display (re_gfx)

The graphics display runs under SunTools. It provides the ability to start up Rover's display window, clear it, and draw crosses. lines and boxes in the window. These facilities are used by the clusters to display their results as Rover runs.

DataCube Interface (re_dq)

This library implements functions to digitize a new image frame in the frame buffer. and to get any subwindow of the frame buffer for closer inspection.

Binary Line Segmenting (rs_lib)

A one-dimensional Kirsch edge detector and general-purpose line segmenter are implemented. The edge detector is a simple "difference of boxes" applied to each point on the line (image vector) that is an argument to the edge detector. Thus the result of the edge detector is a magnitude vector in which rising edges of brightness in the image vector appear as peaks, falling edges appear as valleys, and segments of constant brightness give no response. The edge detector also returns the mean and standard deviation of the brightnesses of the points on the line. The line segmenter locates "peak-valley" pairs whose absolute values exceed a threshold supplied to the function as an argument.

Segment and Blob Lists (re_segbuf)

Essentially, the operations new, insert, and member (like the operations on the abstract data type, set) are implemented for use on segment and blob lists.

Temporary Image Buffers (re_tib)

The operations new, get_image, and free are provided for using the Temporary Image Buffers (TIBs). TIBs are used in the Object Discrimination Cluster to hold subimages from the image frame.

Image Partitioning (re_partition)

Facilities are provided for searching within and splitting image partitions in TIBs.

Object Color Identification (re_color)

This library implements functions to calculate the "color" of an image, and to match a color against the system's current registry of colors. The color an object is the pair (mean, variance) of the brightness of the image of the object. It is assumed to uniquely identify an object.

World Database (DB) Manager (re_world)

This library provides the interface to the spatially indexed world model. It implements the operations necessary initialize the DB, search a spatial region of the world, attempt to match a given object with an existing one in the DB, update an existing object with new information, insert a new object in the DB, and degrade the information in the DB.

References

[Coombs and Marsh 1986] David J. Coombs and Brian D. Marsh, Roving eyes — prototype of an active vision system, December 1986. CSC 400/446 Project Report.

[Kernighan and Plauger 1978] Brian W. Kernighan and P. J. Plauger, *The Elements of Programming Style*, McGraw-Hill Book Company, second edition, 1978.

A Building on Rover

For the reader who wishes to extend the current implementation of Rover to realize greater functionality, this section is devoted to outlining our mistakes for your benefit, and indicating obvious directions for extending Rover.

A.1 Hints for Anguish-free Hacking

We offer these suggestions for your enhanced hacking pleasure (take them with as many grains of salt as you like):

1. Read *The Elements of Programming Style* [Kernighan and Plauger 1978]. At least scan the Summary of Rules at the end—it only takes two minutes, and the reminders may save you many horrors we were not spared in the initial implementation.

2. Follow the conventions established in the existing code. Each lends itself to clear code, and minimum interaction between source code in separate files.

3. Adopt the following convention regarding allocating and freeing memory for temporary results:

   - Any library that exports a function which returns `malloc`'d memory as a result must also export a function to *free* such objects as its functions create.
   - Any function using a function that returns `malloc`'d memory is responsible for utilizing the associated *freeing* mechanism to prevent memory leaks.

4. Of course, it is even more important to avoid `free`'ing memory that other functions may reference in future, as this leads to unpredictable results.

5. Try to use a tight design-implement-test cycle to keep changes as incremental as possible. And of course keep backup copies of the latest version of working code. (RCS is fairly nice for this.)

A.2 Exercises and Obvious Extensions

These exercises are intended as vehicles for familiarizing the reader with the core ideas of Rover and the current implementation. Exercises affecting specific parts of the system appear early in the list, and enhancements involving the entire system later.

1. *(useful)* Do something better with clipped windows than discarding them. For instance, try getting a subimage from the frame buffer adjacent to the clipped edge to grow the window in hopes of finding the whole object. Windows are clipped frequently in practice, so this could lead to significant gains in performance. It could, of course, be argued that the Raster Scanner should be improved to reduce this frequency. (Why not do both?) Also, in the presence of occluded images (consider several balls, each overlapping) we might not be able to afford to discard partial images.

2. *(useful)* Replace simple thresholding in the Object Discrimination Cluster with sparse application of an edge operator to locate approximate boundaries of objects. (The rs_lib was not available when the od_* cluster was implemented.)

3. *(useful)* Develop an error-handling scheme to enable the system to die gracefully and diagnostically.

4. *(tedious)* Apply convention 3 in Section A.1 to the existing code to find and plug memory leaks.

5. *(insidious)* Find the bug that causes floating exceptions occasionally and fix it. Perhaps a local patch will be best, but it may be the result of a previous function not doing its job properly.

6. *(straight-forward)* Extend the declaration of the return values of modules to be a `struct union` and use this to explore top-down strategies for directing the search of the image frame and processing of located blobs.

7. *(tweaking)* Fiddle with the compile-time parameters (*e.g.*confidence thresholds, error measures and limits) of the system to improve performance in any or all modules and libraries. In fact, making these parameters command-line arguments (with defaults, of course) might speed development efforts, although accessing variables slows execution when compared to compiled constants.

8. *(model extension)* Extend the model of the world, and the library functions to recognize occlusion explicitly (rather than dealing with it implicitly as the current implementation does).

9. *(for kicks)* Exploit existing handles (*e.g.*pointers to specific segments in the segment list) and devise your own to improve access efficiency in the system's data structures.

10. *(even better than 9)* Replace simple linked lists with data structures which are more efficient where your analysis indicates improvements will be yielded for the current system or an expanded version(!).

11. *(interesting)* Explore other structures for organizing the system (*e.g.*hierarchical—see Section 1.1 on inter-module communication) and other control strategies (*e.g.*more direction by the executive, dynamic search for blobs, and using dynamically assigned priorities to direct processing).

12. *(expanding universe)* Extend the system to recognize and handle other types of objects (*e.g.*cubes) and more sophisticated identification methods (*e.g.*markings on objects).

13. *(interesting and difficult)* Extend the model and system to explore a larger universe by moving the camera, and maybe even add another camera.

A.3 Templates for Your Own Code

To illustrate the basic forms of the main types of Rover's components, Figure 3 gives a sample of how the skeleton of a cluster declaration file should look. Similarly, Figure 4 describes how a module in that cluster might appear, and Figures 5 and 6 illustrate the essential structure of a library.

B Rover's Code

The *Rover Programmer's Guide* concludes with more detail on the existing source code itself. The code can be found on the system in `/u/coombs/projects/rover`. An executable is available to be run (on the Sun with the DataCube—currently betelgeuse) and is located in directory `/u/coombs/projects/rover/bin`. It is invoked by

```
rover [-c<camera#>] [-f<follow-target#>]
```

from the shell.

B.1 Coding Conventions

Several conventions are followed in the Rover code. Some of the more helpful ones are listed here.

- **Declaration files** are protected against being multiply included (leading to redefinitions of objects and types) by defining a constant upon the first inclusion that acts as a *guard* against subsequent inclusion during a single compilation. (See Figures 3 and 5.)

- **Type definitions** are named in either of two common forms:

  1. `NEW_TYPE` in all capital letters, or
  2. `new_type_t` in lower case, with the suffix "`_t`".

- **Declarations** are made as local as possible to avoid interference among types, data objects, and functions.

- **File name prefixes** refer to the major component of the system to which the file belongs. The three primary prefixes in the current code are:

  1. **re_**—the system at large (Roving Eyes)
  2. **rs_**—the Raster Scan cluster
  3. **od_**—the Object Discrimination cluster.

```
/* Rover Sample Cluster Declaration Template --- for EXPORTING
   declarations to modules that need to know your types, etc to
   interact with this cluster.   */

#ifndef CLUSTER_TYPES                  /* Protect your declaration files from
                                       being included more than once.
                                       Hence, this #define'd constant must
                                       be unique in your system.  That's
                                       why we base it on the file name, as
                                       a convention.  */
#define CLUSTER_TYPES 1

  /* Include only those declarations needed to declare the types, etc
     that you declare here.   */
#include "re_types.h"                   /* in case you need global declarations */
#include "re_queue.h"                   /* needed to declare your modules */



  /* Module parameter types are declared here so other modules can
     construct arguments for your modules.   */
typedef struct {
  int arg1;
  float arg2;
} mod_parm_t;


/* export modules in this cluster so other modules can enqueue them on
   the task queue.   */
extern q_func_t this_module();

#endif CLUSTER_TYPES
```

Figure 3: Sample Cluster Declaration (cluster_types.h)

```
/* Rover Sample Module Template --- Include declarations of global
   system, libraries, and other clusters that this module interacts
   with.   */

#include "re_types.h"              /* Always include rover's global types */
#include "re_exec.h"              /* declaration of global data objects */

#include "re_queue.h"              /* any necessary libraries */
#include "re_gfx.h"

#include "other_cluster_types.h" /* a cluster this module will
                                    interact with */

#include "cluster_types.h"       /* this cluster's declarations */



q_func_t                        /* every module returns type q_func_t
                                   (declared in re_queue.h) */
module(my_parm)
     mod_parm_t * my_parm;
{
  other_mod_parm_t * other_mod_args; /* declared in other_cluster_types.h */
  q_func_t return_value;


  /* my processing here */

  /* enqueue another module on the global task queue (work_q, from
     re_exec.h) to perform the next logical operation based on what I
     have seen.  Coerce the type of the args-ptr to what the queue
     library expects.  */
  enqueue(work_q, other_module, (q_arg_ptr_t) other_mod_args,
        arg_free_fn, OTHER_MOD_PRIO);

  return (q_func_t) return_value; /* return a value */
}
```

Figure 4: Sample Module Source (module.c)

```
 /* Rover Sample Library Declaration Template --- for EXPORTING
    declarations to modules and libraries that want to use the
    facilities you provide.   */

#ifndef LIBRARY                           /* Always protect your declaration
                                      files from be included more
                                      than once. */
#define LIBRARY 1

#include "re_types.h"                   /* any declarations needed */



 /* Library function parameter type declarations */
typedef some_type arg_type1;
typedef global_type arg_type2;         /* global_type declared
                                      in re_types.h */
typedef return_type func1_t;
typedef another_type func2_t;


/* export functions in this library */
extern func1_t func1();
extern func2_t func2();

#endif LIBRARY
```

Figure 5: Sample Library Declaration (library.h)

```
/* Rover Sample Library Template --- this file (library.c) contains
   the library functions advertised in library.h and any internal
   utilities that are expected to be of use only in implementing
   this library's facilities.   */

#include "re_types.h"                /* Always include rover's global types */
#include "re_exec.h"                /* declaration of global data objects
                                   if necessary */

#include "re_queue.h"                /* any necessary other libraries */
#include "re_gfx.h"

#include "library.h"                /* my declarations */



func1_t                               /* lib functions declare their
                                   return types */
func1(arg1, arg2)
     arg_type1 arg1, arg2;
{
  func1_t return_value;

  /* my code here */

  return (func1_t) return_value; /* return my value */
}



func2_t                               /* lib functions declare their
                                   return types */
func2(arg1, arg2)
     arg_type2 arg1, arg2;
{
  func2_t return_value;

  /* my code here */

  return (func2_t) return_value; /* return my value */
}
```

Figure 6: Sample Library Source (library.c)

Appendix B-3

CSC 400/446 Project Report: Roving Eyes —Prototype of an Active Vision System

David J. Coombs
Brian D. Marsh*
University of Rochester Computer Science Department

15 December 1986

Abstract

The Roving Eyes project is an experiment in active vision. We present the design and implementation of a prototype that tracks colored balls in images from an on-line CCD camera. **Rover** is designed to keep up with its rapidly changing environment by handling best and average case conditions and ignoring the worst case. This strategy is predicated on the assumption that worst case conditions will not persist for long periods of time and the system's limited resources should be directed at the problems which are likely to yield the most results for the least effort. This allows Rover's techniques to be less sophisticated and consequently faster. Each of Rover's major functional units is relatively isolated from the others, and an executive which knows all the functional units directs the computation by deciding which jobs would be most effective to run. This organization is realized with a priority queue of jobs and their arguments. Rover's structure not only allows it to adapt its strategy to the environment, but also makes the system extensible. A capability can be added to the system by adding a functional module with a well-defined interface and by modifying the executive to make use of the new module. Possible generalizations and future work are discussed.

*Many thanks are due to Chris Brown, Jerry Feldman and Brian Madden for guiding and encouraging this project, and to Stuart Friedberg for taking the time to profile and speed up our code.

Contents

List of Figures

1 Introduction

The Roving Eyes project (**Rover**) is an experiment in the design and implementation of an active vision system. Such a system is placed in a dynamic world and interacts with it in a non-trivial way. For this project, the interaction is the identification and tracking of moving (as well as stationary) objects. Images are input using a CCD camera mounted on the Department's robot head. Once loaded into a Datacube frame buffer, these images are then transferred into a Sun-2/120 and analyzed to detect areas of motion. These areas are further analyzed to detect the specific identity of the moving objects. The results of this identification process are then incorporated into a database which represents the system's *model* of the world. With such a model, the system is capable of maintaining accurate correspondence between distinct objects over time. This temporal interaction with its real world means that the system must be *real time* in some sense. Hence, Rover represents a "real time" vision system with cognitive as well as sensory abilities.

The initial Rover prototype has been designed only to deal with distinctly colored spheres. This represents a necessary simplification of the cognitive domain intended to facilitate the successful construction of the prototype. Dealing with only spheres permits the development effort to be concentrated on the system as a whole by constraining the amount of sophistication needed to reasonable first-pass limits. Ultimately, Rover will collect enough information to perform such tasks as identifying a solitary block in a field of spheres or identifying particular blocks not just by color (which in fact is just the simple calculation of a moment) but by using alphabet blocks which may be distinguished by the letters on their faces.

The structure of the prototype has been strongly influenced by a contemporary understanding of how the human visual system works. Readings, in particular [Lev85], and close interaction with Brian Madden, a post-doctoral fellow whose particular interest is biological vision systems, provided us with many insights about how best to approach the design problem.

In addition to the participation of Brian Madden, this project comprises our term project for CSC400 and CSC446. It was conducted under the auspices of Chris Brown and in conjunction with the Active Vision Group here at the University of Rochester. That group currently consists of Brian Marsh, Dave Coombs, Barun Chandra, Brian Madden, and Chris Brown. The project itself represents a continuation of work begun by Michael Swain, Chris Brown and Dana Ballard using the robot head to track the centroid of a brightness distribution calculated from a light source moving in front of the cameras.

This document presents the design work and implementation work that has gone into the development of the Rover prototype. It is organized as follows. First, the general problem of maintaining correspondence over time in a changing world is addressed. The next section discusses constraints on our design. The next section

discusses the actual design of the prototype. Specific problems and solutions are discussed. The following section contains a discussion of the actual implementation and structure of the system. The major modules are identified and the various software packages that were constructed are discussed. The last section contains our conclusions about the design and about the implementation. Problem areas that went unaddressed either due to oversight or for the sake of simplicity are identified. Finally, as a technical appendix the graphics display is described and a sample run of the prototype is presented.

The source code for the entire system can be found in `/usr/vision/src/rover/src` The modules are named in the following way:

- **re_** – General purpose routines.
- **od_** – Object discrimination routines.
- **re_rs** – Raster segmentation routines.
- **re_dq** – Datacube interface routines.

2 Maintaining Correspondence - Strategic Issues

One of the major problems in tracking moving objects is the *correspondence problem*. Specifically, given two images, we want to be able to identify those objects in both images which actually represent the same observed object. It raises issues of cognition as well as sensation and is not easily solved by simple template matching techniques. Unlike biological vision systems which can sample at a high enough frequency to avoid it, it is a typical problem facing computer vision systems like Rover.

Biological vision systems avoid the correspondence problem by sampling at a very high rate. Computer vision systems can do the same thing and reduce the severity of the problem by obtaining images of the same objects across very short time intervals. Keeping the sampling time interval short, however, requires Rover's cognitive analysis to be fairly fast, and consequently it must be simple. Due to the simplicity of these algorithms, we can only expect them to succeed most of the time. Occasional failures are tolerable because the world is changing and will likely present more favorable data within a short time.

Rover's strategy for maintaining correspondence between objects in a scene is based on a separation between cognition (and attention) and sensation. Maintaining correspondence over time is a cognitive ability. Motion detection and object identification are lower-level, sensory problems. Sensory techniques (*e.g.* for motion and blob detection) are used to analyze the current image and translate it into symbols that may be effectively manipulated by cognitive processes. These processes use

this information not only to maintain correspondence but also to focus attention on areas of relative importance.

The most natural way to coordinate sensory tasks and cognitive tasks would be in a connectionist event driven semantic network. Such a structure run in parallel would present an extremely powerful organization. Unfortunately, the machine used for the implementation of the prototype is not a parallel architecture. Hence, we use the resources at our disposal as efficiently as possible. This translates to pseudo real-time pseudo parallel tasking. Sub-problems are kept small and are executed only when their relative importance is great enough to the overall functioning of the system. These sub-tasks are:

1. Several kinds of independent sensory analyses continuously process raw image data.

2. Cognitive processes focus (the more expensive) attentive resources (3) on interesting parts of the scene. When they need input from the real world they sample the results from (1).

3. There may be more than one simultaneous locus of attention and more than one attentive process working on each locus at the same time.

Rover's organization is based on this approach to understanding a changing world.

So Rover's behavior can be characterized by a coping strategy—it accumulates as much information as it can at every moment, but it must guard against spending a an excessive amount of time extracting a particular bit of information and consequently losing track of the objects in the scene. For this reason, the system accumulates information incrementally, to preserve as many results as possible if analysis must be cut short during periods of rapid change.

3 Real World Design Constraints

The Rover prototype is constrained by several environmental factors beyond our control. The most important constraint on our design is the computational environment used for the implementation. In particular, the supporting hardware, a Sun-2/120, is a serial machine, with no reasonable facility for exploiting parallelism. With no mechanism for parallel task execution, there is no natural way to realize our task-oriented system organization. At the same time we want our implementation to embody the natural strategy described above. To do this, our system will explicitly multiplex its analysis between the sensory and cognitive levels. The serial nature of the computational resources make it essential to be able to constantly direct our resources at the most promising task. In a parallel environment, irrelevant tasks do not seriously impair the overall computation since other computation

is proceeding concurrently. In a serial environment irrelevant computation can be disasterous for system performance. To enable processing to focus on only the most promising areas, it is essential that our analysis be broken into small computational tasks. At the end of each task the relative importance of that area of analysis can be re-evaluated.

One of the most telling limitations of the hardware is the bottleneck that exists between the Datacube frame buffer and the Sun. When the project began, it took a full 8 seconds to transfer the contents of the entire frame buffer to on-board memory. Due to the lack of powerful image processing hardware all image operations have to be performed on the Sun. Hand optimizations lowered this figure by a factor of 8, but the significance of the bottleneck is still substantial.

Another major consideration was that the prototype design, once implemented, be easily extensible. We feel there is significant potential for future research involving the Rover system and we want to provide a useful software base for this work. The Rover prototype is designed to facilitate the replacement and addition of functional units. This will ease the eventual replacement of the simple routines of the initial prototype with more sophisticated ones. Although we place some credence in the *Waffle Principle*[1] we believe that the framework of our system will provide a useful framework for the solution of active vision problems long after the initial prototype has been thrown out.

Perhaps of equal importance to future research is our expectation that the supporting hardware will be changing fairly soon with the arrival of new Suns and image processing boards. This new equipment will make it possible to do much more sophisticated analysis of images and much more computationally intensive processing of our symbolic representations.

4 The Rover Prototype

This section describes the Rover Prototype. Design goals and issues are outlined. Constraints made to facilitate prototype development are discussed. The framework of the prototype is described along with a discussion of its major modules.

4.1 Design Issues

The Rover prototype both embodies the design strategy discussed previously and resolves many issues that would not have been dealt with but for the actual implementation.

[1]The waffle principle states that the first attempt at an implementation should be thrown out at the second stage and the system should be implemented again from scratch.

4 THE ROVER PROTOTYPE

The principal goal of the Rover system is to maintain correspondence between moving objects in the viewing plane. To do this many different elements need to be manipulated. The world database representing the most current state of the world needs to be maintained. To keep this information up to date, input images must be analyzed to detect areas of motion. These areas of interest in the original image are then correlated with the world database and if necessary have further discriminatory techniques applied to them. Since there are potentially multiple areas of interest in any input image but a limited amount of computational power and time to spend processing them, the cycles spent processing each area must be carefully monitored to insure that the information derived from each input image is complete as possible. It is conceivable that any input image will contain more information than the system can process in a reasonable amount of time. If this happens the image and all associated processing is abandoned for a fresh view of the world.

To perform these various tasks, the prototype is broken up into three main modules:

- **Executive** – Responsible for overall system coordination and task scheduling.
- **Raster Segmentation/Motion Detection** – Responsible for detecting areas of motion in the input image and for segmenting the image into small manageable sub-images.
- **Object Discrimination and Correspondence** – Responsible for identifying the sub-images supplied by the Raster Segmentation module and integrating them into the world database.

The remaining sub-sections in the Prototype section of this report describe these various modules in detail.

Before beginning our discussion of the prototype modules, it is important to note the assumptions we made on the world to facilitate the development of the prototype. Our initial ambitions for the Rover prototype included such things as using alphabet blocks as tracking targets. Since our overriding concern was to complete a working prototype by the end of the term, we constrained the problem Rover would have to solve as follows:[2]

- **Simple Targets** – The targets used for tracking are different colored spheres of uniform reflectance. Actual identification of different spheres is done on the basis of past position and reflectance.
- **Carefully Controlled Lighting** – Shadows are not a problem that with which we wanted to deal in the initial prototype.

[2]Especially since we didn't begin coding until what was effectively the last week of school

- **Limited Cognition** – The amount of correspondence that would be attempted by actual cognition was severly limited.

- **Horizontal Motion Only** – The permitted motion for an observed object is limited to that along the y-axis (horizontal) only. (In fact we were able to loosen this constraint considerably).

- **No Occlusion** – Targets are not permitted to occlude one another. This was another constraint which the design of our system obviated.

- **No Complete Displacement** – Balls are not allowed to swap position.

- **Control Over World** – Random motion is not permitted. In fact, we reserve the right to control motion in the world (say by slowing it down) sufficiently to allow our system to function.

The utility of each assumption, while perhaps not yet clear, is discussed in the context of the prototype modules.

4.2 The Executive

The *Executive* is the framework of the prototype which serves to organize all the other functional modules. Its primary responsibilities are organizational. It coordinates the integration of all the functional units, from the extremely low-level sensation oriented modules that perform pixel operations to the higher level cognitive modules that maintain the world database. The specific functions it performs are controlling task scheduling, task execution, resolving temporally global issues of correspondence and system initialization.

Work is done in the system by enqueuing task requests on a general work queue. If and when there is time to process that particular request then the task is dequeued and the corresponding code is invoked. The framework (*i.e.*, main loop) looks like the following:

- *Initialize work queue*

- *Initialize world database*

- *Enqueue initial Raster Segmentation / Motion Detection task*

- *Forever*

  - *Get a task off of the work queue if it isn't empty.*
  - *If the queue is empty then enqueue a Raster Segmentation task and reset the interval timer.*

- *Invoke the routine specified by the task just dequeued.*[3] *If the specified task involves bringing in a completely new image then increment the virtual time stamp counter and reset the interval timer.*
- *If there is any time left in the current interval then return to the work queue for more work. If there is no time left in the current interval then queue a Raster Segmentation task and reset the interval timer.*

The code implementing this framework can be found in `re_exec.c`.

4.2.1 Tasking

For task scheduling the executive uses a simple priority queue of queues. (Implementation is provided by the package in `queue.c`). Tasks may be queued with priorities ranging from 1 upwards where 1 represents the highest priority. When the current task completes execution, the executive will dequeue the highest priority task and execute it. Should there be no tasks waiting for execution then the executive will move on to the next image (we assume that there is always new information that can be input to the system by processing new images).

Tasks are created by enqueuing requests for their execution on the Executive's work queue. The interface to the queue package permits the enqueuing of an operation type, an argument to the specific operation, and the type of the argument to be queued at a specific priority. This was designed to facilitate the future addition of other other types of tasks. It also structures the flow of the system in such a way that would map reasonable to a parallel machine. The actual command for the enqueuing operation is

```
enqueue(work_q, task_type, arg_ptr, arg_type, priority)
Q_HEAD * workq;
short task_type;
baddr_t arg_ptr;
short arg_type;
short priority;
```

The tasks that execute in the current prototype are specified in `re_exec.h`. They are specified by the #defines:

```
/* Operation types */
#define OP_RAS        1
#define OP_OBJ_DIS    2
#define OP_OBJ_COLOR  3
```

[3] This is where the majority of the processing is done. These routines are described in the sections on Raster Segmentation / Motion Detection and Object Discrimination / Correspondence

OP_RAS is used to enqueue a Raster Segmentation / Motion Detection task. No arguments are enqueued. OP_OBJ_DIS is used to enqueue an Object Discrimination task. These tasks are used on images that have just been analyzed by the Raster Segmenter. The image is the argument queued with the task type. This is the first pass of the object discrimination process and is used to insure that the input image is usable and that it contains only one object. OP_OBJ_COLOR is used to enqueue tasks that perform identification on images that have been passed on by Object Discrimination tasks. The actual image serves as the argument and is guaranteed to contain an object by the Object Discrimination tasks.

The priorities at which these tasks are enqueued are:

```
/* Operation priorities */
#define OP_OBJ_COLOR_PRIO 1
#define OP_OBJ_DIS_PRIO   2
#define OP_RAS_PRIO        3
```

These priorities are static, but their assignment is not strictly arbitrary. The Raster Segmentation priority (OP_RAS_PRIO) is the lowest because a raster segmentation task should only be executed when all other processing on other portions of the input image has finished. In the initial prototype this actually represents a degenerate usage of the priority queue since raster segmentation is done only when the work queue empties itself or has been flushed. Regardless of this however, since object discrimination tasks are not queued until the raster segmentation of the current image has been done, the object discrimination routines naturally receive a higher priority since their existence on the queue implies that enough information has been secured for their successful execution. With this in mind, individual object discrimination (OP_OBJ_COLOR) is given priority over the preliminary processing of images (OP_OBJ_DIS) since the queuing of OP_OBJ_COLOR tasks implies that the system is quite close to the complete classification of an object.

It is expected that future work on Rover will encorporate a dynamic priority scheme that takes into account such issues as spatial relevance and confidence levels (see Object Discrimination - Maintaining the World).

4.2.2 Temporal Model of the World

The executive maintains several notions of time. It helps maintain the world database over the course of input images by performing functions not rightly relegated to either of the lower level modules. It also attempts to keep track of the passage of real time in relation to the processing it controls. Should excessive computational effort be exerted in the processing of any one image the model of the world maintained internally could fall hopelessly out-of-date.

The executive manipulates various lower-level modules (*i.e.*, Raster Segmentation and Object Discrimination) to perform the analysis of an image that represents

a snapshot of the world at a given moment. These routines perform an analysis that is largely restricted to the input snapshot and the past history of the world stored in the world database. An obvious problem with this is that information that is either missing in the current snapshot of the world or that is simply missed by the analysis should still be accounted for in some way. An obvious example occurs when an object moves from the field of view. While it would be reasonable to simply delete the object from the world database, this makes the database extremely volatile since a mistake by the low level routines at any point could result in the accidental but erroneous removal of an object. To counter this problem the system maintains a measure of confidence in each object stored in the world database. Whenever the position and identification of a particular object are reaffirmed, this confidence is raised to a maximum level. Should an image be processed without any new information being provided about an object, for whatever the reason, the confidence in the identity (as well as its existence) is lowered.[4] Once this confidence falls below a certain threshhold the object is deleted. This provides the system with some measure of resiliency.

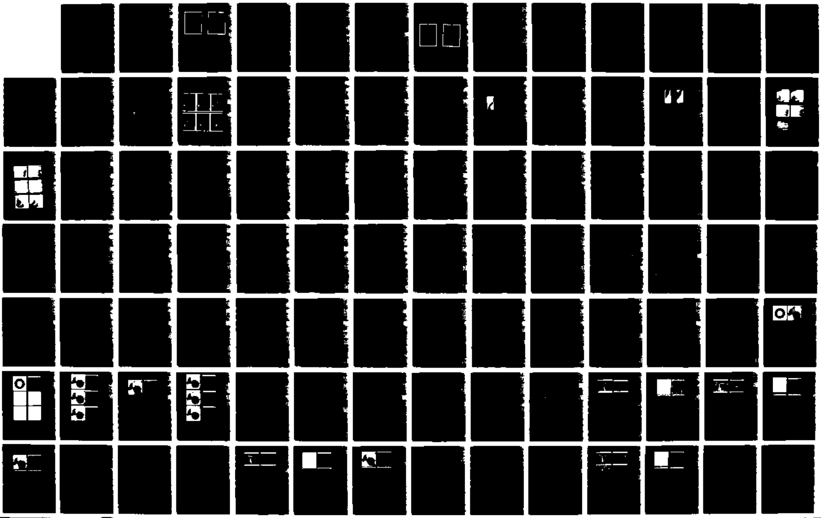
An additional interesting result of this degradation of objects which receive no updating over a period of time is that the system is capable of dealing with temporary occlusion. In such a case, two objects will enter a spatial relationship such that their images overlap resulting in that partial occlusion of one of them. Should they be identified as a single object by the Raster Segmentation and Object Discrimination procedures then one of two things can happen. The combined colors of the objects might be identified in a new color and assigned a completely new entry (albeit a *fake* one) in the world database. Alternatively, the object will be identified as one of the existing objects that the system expects to find in the area. In this case there will be a partially erroneous aliasing of one of the objects to the other. Since we assume that the occlusion is temporary, in the first case, the objects will separate and the confidence in the *fake* object will eventually degrade and it will be removed. Clearly no harm is done here. In the second case as long as the objects separate soon enough the world model will still contain the aliased object and its position can be updated. Again, the loss of correspondence is minimized.

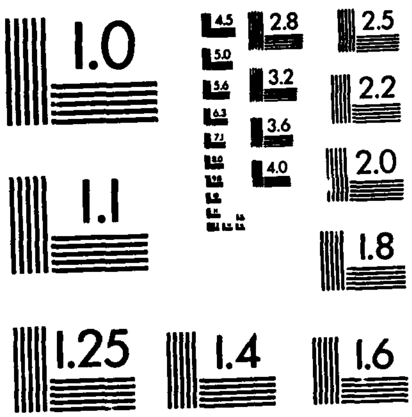
The executive maintains a notion of a virtual time segment (enforced by the global variable `current_vts`) whose length is defined to be `VTS_LENGTH` seconds. This segment is the maximum length of time that Rover should spend processing any single input image. If at the end of a time segment there is still processing to do on the current image, it is likely that any information that could be derived from it would be obtained at the expense of falling way behind the state of the real world. To avoid this problem, any tasks that are pending at this point are destroyed and processing on the new image is started by enqueuing a new Raster

[4] In the implementation this operation is performed just prior to the processing of a new image by the routine `world_degrade()`.

AD-A190 597 NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT 1986 VOLUME 8..(U) SYRACUSE UNIV NY C M BROWN JUN 88 RADC-TR-88-11-VOL-8-PT-B F30602-85-C-0008 2/3

UNCLASSIFIED F/G 12/9 NL




1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

Segmentation task. Task destruction is accomplished by simply flushing the queue of any waiting tasks. Additionally though, all objects whose internal representation went un-updated have their confidence degraded.

The virtual time segment is also used for manipulating the world database. When a database object is updated it is stamped with the value in current_vts. This value is then used to exempt the updated object from further processing in this time segment.

4.2.3 System Initialization

System initialization consists of initializing the various system data structures. These are:

- **World Database** - This is a spatially indexed representation of the world (see the **Data Structures** section).
- **Work Queue** - This is the priority queue used for task scheduling and execution.
- **Graphics Screen** - This is the window used on the Sun screen for demonstrating the actions being performed by Rover.

The various interfaces for the software packages used to manipulate these data structures can be found in the **Data Structures** section.

Additionally, the initial Raster Segmentation tasks are enqueued and the main loop is entered.

4.3 Image Segmentation – Divining Objects

The Image Segmentation module isolates the locations of objects in the scene using a coarse sampling of the image. When the executive notices that a potential object has been located, it enqueues an Object Discriminater to examine it. Thus, the Image Segmenter is a cheap filter that saves the expense of inputting uninteresting portions of the image from the frame buffer (a real bottleneck) and allows the system to concentrate high resolution (expensive) operations on areas of the image which are likely to produce the most useful results.

The Image Segmenter's design was strongly influenced by the characteristics of the Datacube frame buffer and our interface to it. Grabbing horizontal lines from the frame buffer is faster than any other mode of acquisition, so the Image Segmenter uses a coarse sampling of horizontal "rasters" from the frame buffer to locate potential objects.

Each raster is examined by an image segmenting task. (Each "frame" in the image sequence consists of a pair of images taken in rapid succession from the CCD

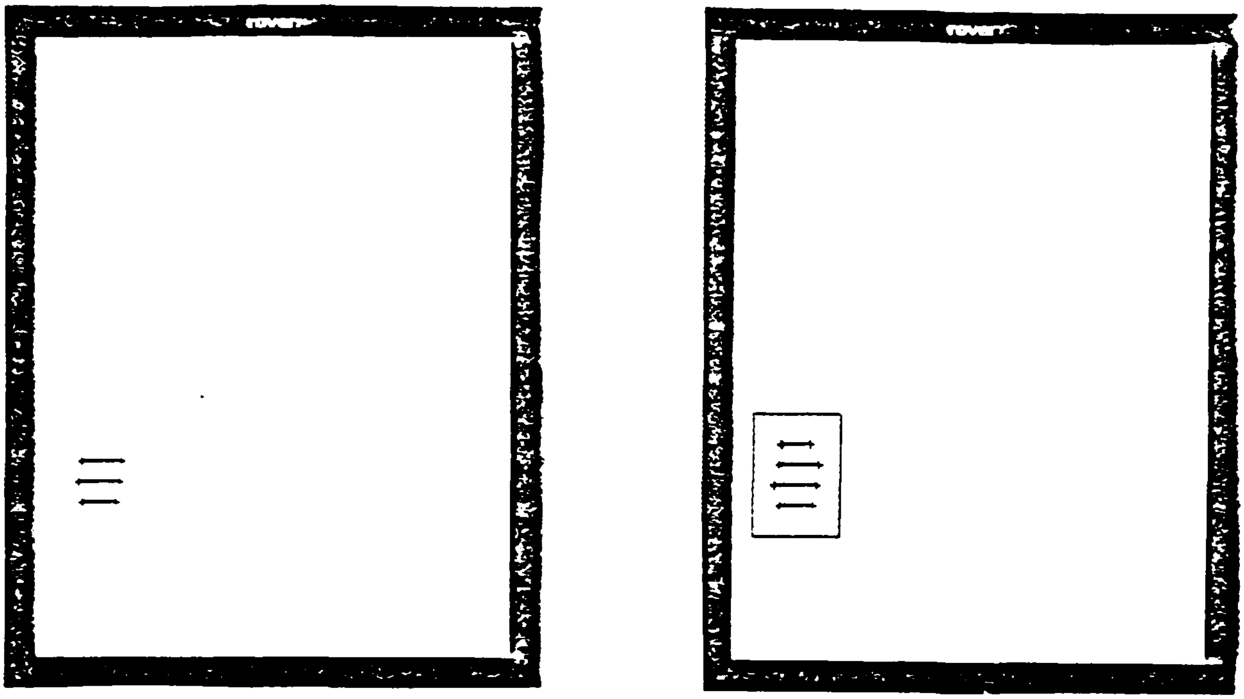


Figure 1: Paired raster segments and an object region grown from them. Paired segments are denoted by crosses and the object region is enclosed in a box.

camera. These images are digitized in the left and the right halves of the frame buffer, respectively.) At the beginning of processing each frame, the executive enqueues a raster segmenting task for each of the rasters that will be scanned in the frame.

Each image segmenter performs these operations on its raster:

1. **Segment the raster**—locate light segments of the raster which might be caused by a light object on the dark field of the background.

2. **Identify pairs of segments**—match pairs of raster segments which appear in both the first and second images and call these *positive* segments (*i.e.* not likely due to noise in the image).

3. **Identify vertically associated rasters**—grow "regions" of *positive* raster segments which seem to indicate a single object.

Figure 1 demonstrates the results of segment pairing and object growing.

4.3.1 Segmenting Rasters

The raster segmenter uses a one-dimensional edge operator (based on the Kirsch operator) to detect high-contrast edges in its raster from the image. A *positive* segment consists of a *peak-valley pair* in the response of the edge detector. Such a pair should indicate the presence of a light-colored object in the image (against the dark background). The segmenter uses a threshold function (of the mean and variance of the intensity of the image raster) to determine what response magnitude constitutes a positive response of the edge operator (rather than noise).

4.3.2 Pairing Segments

Segment pairs of a raster match in the segment in the first image overlaps the segment in the second image. Any unpaired segments are ignored as bogus or noisy false responses. This pairing criterion effectively limits the class of objects the system will recognize to those whose retinal image is not completely displaced in "a blink of the eye."

Thus, a small object traveling at high velocity perpendicular to the line of sight will be ignored because its images in the first and second blinks will not correspond to each other. Conversely, an object of virtually any size speeding directly at the camera will get a very strong match.

4.3.3 Growing Objects

The executive maintains a list of the responses from the image segmenting tasks (one for each raster being scanned). As each segmenting task returns with its list of segment pairs, it updates the record for its raster. In addition, the executive keeps a list of object regions. As each segmenter returns its list of segments, it tries to update the object list with the new information it is returning about the image. If one of the segments the task found corresponds to one of the objects in the list that segment is added to the object. (A segment corresponds to an object if it is adjacent to the top or bottom of the object, and it overlaps with any of the rasters already in the object, and its velocity—which is estimated from the rapid pair of images—matches closely the velocity of the object.) If the segment matches no current objects, the list of segments is searched for an adjacent (unmatched) segment which could form a new object with the segment being returned. If a segmenter task returns a negative response (no segments found) the object list is searched for objects which are adjacent to the empty raster being returned. Any such objects are bounded by this raster and hence complete. Of course, if a segmenter misses a segment, it would clip' the object that it failed to see. An object is also bounded if the adjacent raster contains no segments which match the object (although there

may be some segments in the raster). The executive, rather than the segmenting tasks, detects this condition.

When the executive notices that an object is bounded above and below, it enqueues an object discrimination task to examine the object and try to match it against the current model of the world, as described elsewhere in this report.

4.4 Object Discrimination - Maintaining Correspondence

Once areas of movement have been identified by the tracking module, the identification of individual objects must be determined. The initial identification and maintenance of correspondence with particular objects is done by the Object Discrimination module. This module consists of high level routines used to interface with the executive and low level routines used exclusively by the module for processing raw pixel data.

When the tracking module locates an area in which it believes there is a moving object, a request for closer inspection is put in the system queue and an entry is made to a Temporary Image Buffer (TIB). The queued request will contain a reference to this buffer and will be used when the request is serviced by the Object Discrimination Module (this one). Note that no part of the image is actually copied out of the frame buffer at this point.

The analysis performed by this module is broken into the following sections:

- **Image Validity** - `do_obj_dis()` Insures that the sub-image returned by Raster Segmentation is useful
  - **Detection of Good Images** - Determines if part of the object in the target image has been cut off by the border of the sub-window.
  - **Detection of Partitionable Images** - Determines if the sub-window contains more than one object in it.
  - **Partitioning of Images** - Actually partitions the sub-window so that each partition contains only one image.
- **Maintaining the world** - `do_one_obj()`) Actually does the high level correspondence that allows the world database to reflect the state of the observed world at any point in time. This information is used to determine how much low level processing is necessary.
  - **Spatial Matching** - Attempts to do object identification by correlating object position with predictions about the way the world will look given the passage of time and its effect on the world database.

- **Color Identification** - Expresses the *color* of the object in the input window as the double containing the mean and variance of the object image intensity.
- **Color Matching** - Determines if the *color* is one already seen by the system. Assigns each color a unique integer identifier.

The first stage is implemented by the routine `do_obj_dis` and the second stage is implemented by the routine `do_one_obj`. These routines can be found in `od_discrim.c`. Low level support routines can be found in `od_obj_dis.c`.

4.4.1 Image Validity

The first stage of processing images received from the raster segmentation module is determining the potential usefulness of the images. In our view, complete identification of an image is a computationally expensive procedure. As a result, it makes sense to attempt to determine whether the information derived from a particular image will be of any use.

In this first stage (the routine `do_obj_dis`, the dimensions of the window of interest identified by the raster segmenter are extracted from the `TIB_OBJ` that is passed as a parameter to `do_obj_dis`.[5] An example of this can be seen in Figure 2.

At this point, the image is transferred from the Datacube to Sun memory. The raster segmenter constrains the size of the window considerably.[6] The image is then subjected to a series of tests: determining if an object straddles an edge of the window and determining if multiple objects are present in the window.

The routine `ok()` determines if an object straddles an edge(s) of the window. It does this by passing a simple threshhold operator over each boundary to a depth of `MAX_BORDER_WIDTH` pixels. If a high intensity patch is located, this is taken to indicate that an object is so close to the edge of the window that part of it has been lost. This missing portion may be critical to the correct identification of the object. As a result, we adopt the simple strategy of abandoning further evaluation of this image. However, `ok()` does return a struct containing a list of the boundaries of the image on which the object is incident. This information is intended for use in stretching the window to obtain the missing information. This stretching was not implemented in the prototype.

Once the image has been inspected for incident objects, it is next inspected to determine if there are multiple objects present. This is done by simply scanning horizontally along the image every `B_WIDTH` rows. When a high intensity patch is

[5] The raster segmenter avoids reading blocks of the image from the Datacube frame buffer as the transfer between Datacube and Sun memory is an extremely time consuming operation.

[6] See figures of sample output.

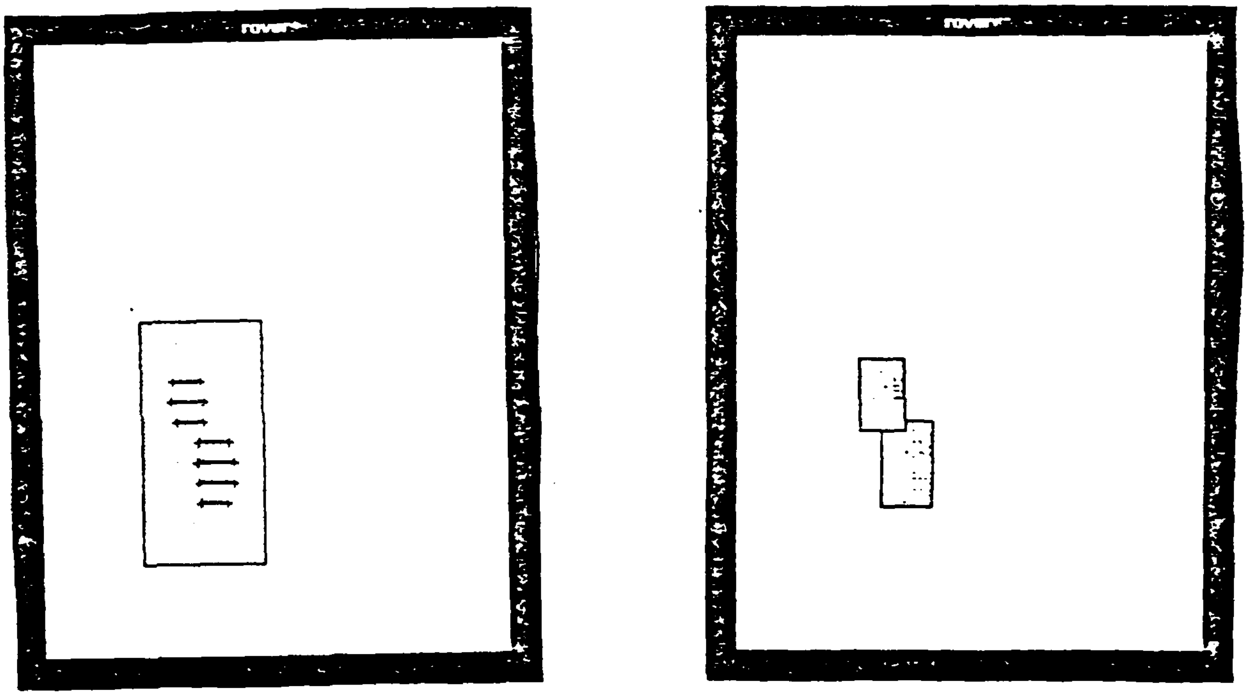


Figure 2: A blob identified by the Image Segmenter and properly split up by the Object Discriminater.

detected, this is considered to be an object. The center of the high intensity patch is calculated and then a line is drawn vertically to determine the upper and lower boundaries of the object. From this information, a more tightly constrained window is drawn around the object. This scanning procedure is continued until the entire input window has been traversed (this is the routine `find_mult()`). A list of object dimensions are returned. An example of this procedure can be seen in Figure 2.

The techniques used for the object location and window partitioning are admittedly primitive. Edge finding, for example, is done by simple threshholding. The prototype does however, use a more sophisticated one dimensional Kirsch operator in the raster segmentation routines. The development of the Kirsch occured concurrent to the development of the object discrimination routines and hence the immediately available threshholding was chosen simply to speed the development effort.

If multiple objects are detected in an invocation of `do_obj_dis` then each object returned by the partitioning objects are placed on the task queue with a request for `do_one_obj`. If only one object is detected then `do_one_obj` is called directly. The rationale behind this is that even though partitioning on the window was completed successfully, the window was of less than perfect quality. Hence, the new sub-windows are assigned a slightly diminished priority by being placed back on the task queue.

4.4.2 Maintaining the World

The second phase of the object discrimination process is responsible not only for the accurate identification of objects at a moment in time, but for doing correspondence over time. This is where the majority of the maintenance is done on the model of the world used to represent what Rover has seen. This involves doing spatial matching to minimize the search space for correspondence; actually identifying objects by color if necessary; and using the spatial and color information to determine if the object identified is an object that has moved and for which correspondence is being maintained, or if that object is new. The appropriate action is then taken on the world model. Appendix A shows an example of a such a correspondence over time.

These entries have information about the last determined position of an item and its last known velocity. Using this information it is possible to predict at a given point in time the new position of the object and its approximate velocity. Should the old velocity of an object and the apparent velocity of the object in the window be somewhat different, it may mean that the objects are different. Alternatively, it may be that the object has changed its velocity. If this is the case then the expected current position with the new velocity is computed and used as a basis for comparison. If the position of the referred window is close to the predicted position of an object then the position of the object in the World is updated and the

request is finished. This "dead-reckoning" approach provides a way of conserving computational power, but provides a potentially less accurate picture of the world. To take this into account, a confidence measure is associated with each object in the World database. Every time the position of an object is updated without actually calculating the identity of the object, i.e. with a position correlation, this measure of confidence is degraded. Once it falls below a certain point, then even if there is a high correlation between the expected position of the object and the position of an unidentified object, the unidentified object will be identified completely.

Once the object is identified either by position or by actual identity, the appropriate World database entry is updated. The current position of the object and its new velocity are recorded. If the identification is by position then the confidence in that identity is degraded. If the identification is by identity the confidence in the identity of the object is set to 100%. If no entry currently exists in the World database, then a new entry is made.

Color is distinguished as the mean and variance of the intensity in the object. The spheres that serve as our initial targets have the nice properties of being relatively invariant in reflectance regardless of the viewing perspective. This simplifies identification considerably. To ease the cognitive burden, we assume that all spheres have a distinct color in Rover's world.

In order to calculate the mean and variance of the object body, simple segmentation is done using the data from the application of an edge finder to the image. The image is scanned row by row. When an edge gradient of significant magnitude is encountered, intensity values are accumulated. Once a following edge gradient is encountered, processing skips to the next row.

Inevitably there are some difference between the mean and variance of two different images of the same-colored sphere. To deal with this problem, we adopt the *ad hoc* approach of considering two *colors* to be the same if they differ by only a certain (magic) percentage. This percentage is the difference of the weighted sum of the means and variances. If the colors are considered to be the same then the object being identified is assigned an integer identifier associated with that color. This identifier is displayed in the figures as a means of demonstrating correspondence and tracking. Should the difference in the two colors be great enough (larger than the magic threshhold function) then the system considers the color to be distinct and assigns it a new identifier.

5 Future Directions

The following section describes work which represents the next logical steps in the full development of the Rover system. It moves Rover closer toward meaningful cognitive interaction with its environment and attempts to solve some of the more basic

sensory problems that are unaddressed in the initial prototype. Typical cognitive development involves using the object history and models of object behavior to aid in identification and correspondence. Typical of these sensory problems are dealing with occlusion and recognizing more complex objects such as alphabet blocks.

5.1 Cognition

Future cognitive developments would involve things such as using the history of the world to guide the identification procedure. The executive can make a guess about what the object in a particular window will be and use it to influence which routines in the Object Discrimination module are invoked. The result will be a much more intelligent guess about how to go about processing image windows.

In addition, the executive could use such information to decide which areas of the scene are likely to contain useful information, ignoring other portions during a crucial detail acquisition task.

Conversely, a good clean image of an object could be saved for later processing if the system must devote its attention to keeping up with a scene which is changing extremely rapidly for a short period of time. As long at the old image can reliably be attributed to the correct object, it can be processed at the system's leisure.

5.2 Dealing with Occlusion

Relaxing the constraints on object movement in Rover's world is an eventual goal. If objects are no longer restricted to non-intersecting horizontal locii then occlusion becomes an extremely important problem to address.

It happens that for certain situations in which occlusion occurs, Rover already maintains correspondence properly. This is done by maintaining a measure of confidence in the existance of an object over time. Should an object stored in the database go without periodic reaffimation of its existance, its confidence will drop below a threshhold and will result in the deletion of the object from the database. Since occluding objects are usually considered by the system to be a completely new object, independent of the separate component objects. Once the objects separate, there will be no further information about this object and hence it will eventually disappear. This technique for dealing with occlusion is ad hoc at best.

A more effective technique would be to deal actively with images in which occlusion occurs. In a rudimentary form this would involve merely detecting the occlusion and eliminating all further processing of the image. This would prevent the appearance of superfluous objects in the world database. A more sophisticated solution would attempt to do object identification in spite of the occlusion.

The problem of dealing with occlusion is thus reduced to detecting it. The

difficulty of detection is proportional to the complexity of the objects which Rover is required to recognize.

When the cognitive domain is distinctly colored spheres, we need only to use a segmentation scheme that detects not only binary changes in intensity using edge operators, but intensity changes from one color to another.

When the cognitive domain becomes more complex, say by using alphabet blocks, or using both the blocks and the spheres, the solution involves using the straight line detection techniques of [BHR86] to detect the outlines of the boxes. General areas of interest would be detected using a simple edge operator with region growing. Connected lines could then be grown using the perimeter of the binary region as a starting point. It would then be necessary to define legal relationships between these lines and base the detection decision on an evaluation of these relationships.

An alternative and potentially more successful approach involves simply assuming that there is only one object if multiple objects are not detected. That is, we ignore occlusion. Instead, we attempt to identify the block using the methods described below. One of the constraints mentioned below is that the figures (letters) that may appear on the blocks are coded beforehand, giving Rover a benchmark against which to compare potential identifications. If the regions in the object can be matched to one of these benchmark id's then the information is used. The gamble is that no information will be derivable if the images are occluding. Empirical results will be needed to determine whether the occlusion will make identification so difficult as to justify the expense (both in computation and development) of good techniques for detecting the occlusion so as to prevent the useless expenditure of computation time.

5.3 Dealing with Blocks

Identification and tracking of alphabet blocks is significantly more complex than the corresponding identification problem with the multi-colored spheres. Although we constrain the problem somewhat by requiring that the same letter occur on every visible face of the blocks, issues of rotation, projective distortion, and actual character recognition must all be resolved. The solutions employed must be fast and effective. They need not, however, work all the time. As long as characters can be effectively recognized most of the time when given a good view, enough information can be gathered to maintain correspondence.

The proposed solution to this recognition problem has three parts:

- Edge Detection
- Segmentation/Blob Growing
- Blob Relations

and is strongly reminiscient of Constructive Solid Geometry Techniques [BB82].

The edge detection facilitates the segmentation/blob growing. The segments are then partitioned by faces and the region described by each face is compared to a dictionary of face relations that describe, in a slant-invariant and rotation-invariant form, the various characters that Rover knows about.

To do segmentation we use a linear region growing approach similar to the one used by the current prototype to grow raster segments into "objects" (see the section on **Growing Objects**). This allows us to segment the image into blobs into time that is linearly proportional to the size of the image. The algorithm uses a collapsing union-find technique to achieve this kind of speed. The segmentation parameterizes the blobs by size and center of mass.

As the blobs in the image are being gathered, relationships between blobs are being recorded. Information such as which blobs are next to which are stored in the blob identifiers assigned by the segmentation process.

Once the image has been segmented into blobs, the blob identifiers are inspected and fully related to one another. This process yields relationships such as which blobs surround which, how many blobs are contained in a particular blob and how many blobs are adjacent to a particular blob. This information should provide a way of quickly discriminating between different characters as long as the blob relationships describing them are significantly different. An obvious example is the difference between X's and O's. The X has a one region containing another, the O a region containing a region containing yet another region. The amount of information stored is small and is easily compared against a dictionary of known recognizable figures.

This technique has advantages over moments [Alt62] in that it seems to be considerably faster and considerably more stable. ([Alt62] advises the calculation of several moments for accurate character identification.) The blob technique provides a means for identifying most distinct figures (it would doubtless have trouble distinguishing between "I" and "l") that is completely invariant with regard to rotation about the z-axis. We would also argue that it is more invariant with regard to the skewing that results when the cubes rotate about their vertical axis.

Of course, due to rotation about the y-axis (vertical), there will be times when the view of the target block is such that the images of the characters on the sides of the target are distorted beyond recognition. Since humans would likely have a difficult time identifying the target, it is certainly reasonable that our algorithm not work all the time. Moreover, if the image is bad enough, it may not be worth the computational effort to extract the identification using any technique.

The effectiveness of this algorithm hinges on the ability to do consistent segmentation. If the segmenter produces different blobs and blob relations when the image is perturbed at all then the technique will fail miserably. We can control Rover's environment somewhat to ease the burden on the segmentation routine, but only

experimentation with real images will indicate whether this is a fruitful approach.

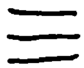
Appendix A

Graphics Display
and
Sample Run

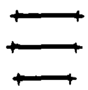
6 Appendix A - Graphics Display and Sample Run

To demonstrate its functionality, the Rover prototype has a simple graphics display that uses the Sun window system. The objects that may appear in the display are:

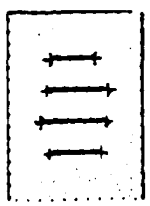
- **Rasters** – Indicate where the raster segmenter thinks an object might be. Looks like:



- **Matched Rasters** – These are raster segments that have been matched between the two halves of the frame buffer. Looks like:

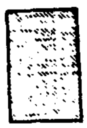


- **Raster Windows** – A window the raster segmentation routines think encompass an object. Looks like:



6 APPENDIX A - GRAPHICS DISPLAY AND SAMPLE RUN

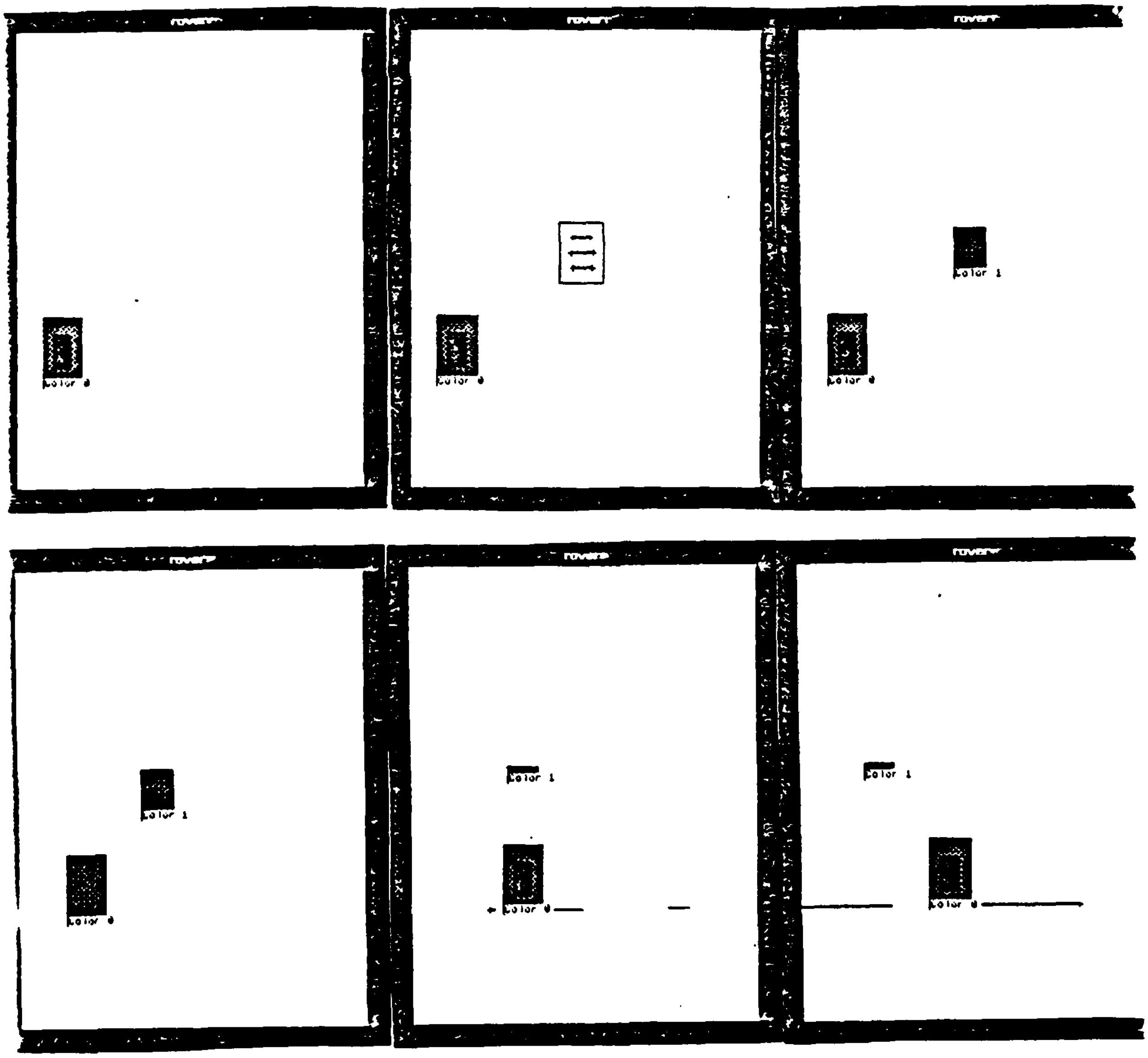
- **Acquisition Windows** - A window the object discrimination routines think will be positively identified as a sphere. Tends to be a tighter fit than the Raster Windows. Looks like:



- **Confirmation Windows** - A window that the object discrimination routines have positively identified as being a sphere of a particular color. The color identifier is printed out in the lower left hand corner of the window. Looks like:

6 APPENDIX A - GRAPHICS DISPLAY AND SAMPLE RUN

The following sequence of figures demonstrates Rover tracking two balls as they cross the field of view, one moving to the right, the other to the left.



References

[Alt62] F. L. Alt. Digital pattern recognition by moments. *Journal of the ACM*, 9:240–258, 1962.

[BB82] Christopher Brown and Dana Ballard. *Computer Vision*. Prentice-Hall, Inc., 1982.

[BHR86] J. B. Burns, A. R. Hanson, and E. M. Riseman. Extracting straight lines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:425–455, 1986.

[Lev85] M. D. Levine. *Vision in Man and Machine*. McGraw-Hill, 1985.

Appendix B-4

THE AUTOMATIC GENERATION OF DIGITAL TERRAIN MODELS FROM SATELLITE IMAGES BY STEREO

Paul R. Cooper
Daniel E. Friedmann
Scott A. Wood

MacDonald, Dettwiler, and Associates
3751 Shell Road
Richmond, B.C.
Canada

INTRODUCTION

Satellites can provide cost effective remotely sensed images. The usefulness of these images is increasing, as the sensors improve with each new satellite. Furthermore, satellite-acquired imagery is in digital form, suggesting the possibility of automating remote sensing tasks such as mapping. To date however, satellite imagery has only yielded two dimensional planimetric information.

With stereo pairs of satellite imagery the capability for generating the third dimension, height, exists as well. The French SPOT satellite (Chevrel, Courtois and Weill, 1981) for example, can image high resolution stereo pairs. Depths are generated from such stereo pairs by stereo matching, normally the task of the human stereo vision system. This paper describes a computer system which automates the process of stereo matching. With this system, the digital equivalent of a contour map (a digital terrain model, or DTM) can be generated automatically directly from digital satellite images.

Digital terrain models are quite useful, serving every purpose that a contour map does, and others as well. For example, terrain dependent parameters such as volumes can be computed easily from DTMs, and DTMs are used in the production of orthophotos. Unfortunately, generating a DTM manually requires many hours. This cost has motivated many attempts to automatically correlate stereo images, without much success.

Recent computational vision research has reported some progress in this area (Barnard and Fischler, 1982). This paper describes work that extends these results and applies them to digital satellite images. The result is a system for generating DTMs with significant capability, demonstrated with results from both simulated SPOT images and real Landsat 5 Thematic Mapper (TM) images. DTMs with height accuracies better than 60 metres have been obtained from Landsat images. Before these results are presented, the problem is discussed in detail, and the algorithm which has been developed is described.

THE PROBLEM AND SOLUTION STRATEGIES

Stereo Parallax Formation

Depth is recovered through stereo by combining two views of the same scene in the world, each acquired from a different viewpoint. Figure 1 outlines the basic situation. Depth at a point in the world causes variation in the position of the image of the point, separately in both views. Depth is recovered by matching such image points, measuring their combined position variation (called stereo parallax or disparity), and converting this parallax to depth.

The Stereo Matching Problem

By far the hardest part of the problem is the matching stage. Matching the two images really implies that the radiometric intensity data from one image, representing a particular piece of the real world, must be matched to intensity data from the second image, representing the same

piece of the real world. Note that this implies considerably more than just matching image intensity data (correlation), because the same piece of the world may look considerably different radiometrically from different points of view, or at different times. Instead, we detect edges in the image, which reflect the true structure of the world, and match these (Marr and Poggio, 1979; Baker and Binford, 1981).

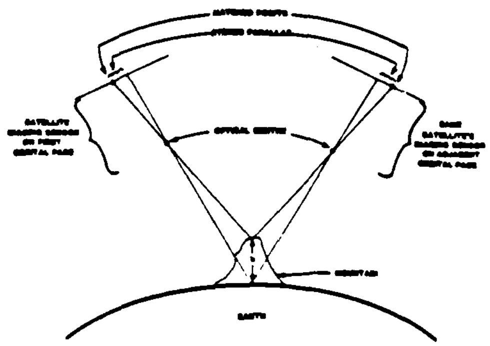


Fig. 1. Simplified satellite stereo imaging

Two sources account for the remaining problems in stereo matching: classic signal-to-noise considerations in the imagery, and the local nature of the depth-induced parallax or position variation. Different depths in the scene cause locally differing amounts of parallax everywhere in both images. A match search technique such as correlation is thus necessarily local in nature. Such local searches suffer "keyhole" effects from lack of global information. Two strategies are adopted to counter these problems. First, connected edges (which we call boundaries) with extent over significant portions of the image are matched. These boundaries are non-local in nature, countering keyhole effects. Secondly, the boundaries are detected and matched in a hierarchic coarse-to-fine fashion. This helps remove the potential for localized match ambiguity, and improves signal-to-noise considerations.

The Stereo Geometry Problem

Once the images are matched, obtaining depth requires a model relating the imaging geometry and measured parallax to the required heights. While the basic model is simply a matter of trigonometry, a variety of complications arise in the satellite image domain. One set of difficulties arises because the imaging geometry varies with time and is difficult to determine accurately. The typical situation in the satellite case is a line scanning sensor mounted on a platform in continuous motion relative to the ground. Appropriate use of an image correction system yields both imaging geometry models and imagery of sufficient quality for matching.

The geometric model must also compensate for problems inherent in side-to-side satellite stereo imaging, where the two images are acquired independently from adjacent orbit paths. Such problems include relative rotation between the images, earth curvature, and perspective distortion effects in SPOT imagery (due to the side-looking angle of the sensor, relative to vertical or nadir looking).

THE SYSTEM

The overall structure of the system is shown schematically in Fig. 2. Each major stage is described below. The input to the system consists of a pair of raw digital stereo images of a scene. The output is both a feature DTM and a gridded DTM. The feature DTM consists of heights everywhere in the scene there are edges or features, such as depth discontinuities. The gridded DTM is evenly spaced height samples representing the entire terrain surface. All heights are in standard format, measured relative to the earth reference geoid.

Image Preparation

In this step, the raw input data is geometrically corrected and prepared for stereo processing. Geometric correction is done with MacDonald Dettwiler's Geocoded Image Correction System (Friedmann, 1981; Friedmann and colleagues, 1983). This removes imagery variation due to sensor and platform motion. The imagery is correc‿d to "precision" level; that is, ground

truth in the form of ground control point coordinates is used to derive highly accurate spacecraft orbit, attitude and imaging models. Usually, such corrected imagery is then geocoded - transformed into some map projection and resampled. In the stereo case however, geocoding is not performed. Instead, the imagery is resampled into an ideal spacecraft dependent projection.

The images are kept in spacecraft dependent projection primarily because they must be put into vertical registration, with corresponding scan lines aligned - so-called epipolar registration. Epipolar lines are defined as the corresponding lines along which stereo matches occur (Barnard and Fischler, 1982), and are equivalent to the scan lines in satellite imagery. This is because depth induced parallax can develop only along the scan line when each is scanned separately, as is the case with satellite imagery. The net result is that both matching and parallax measurement must occur between corresponding scan lines.

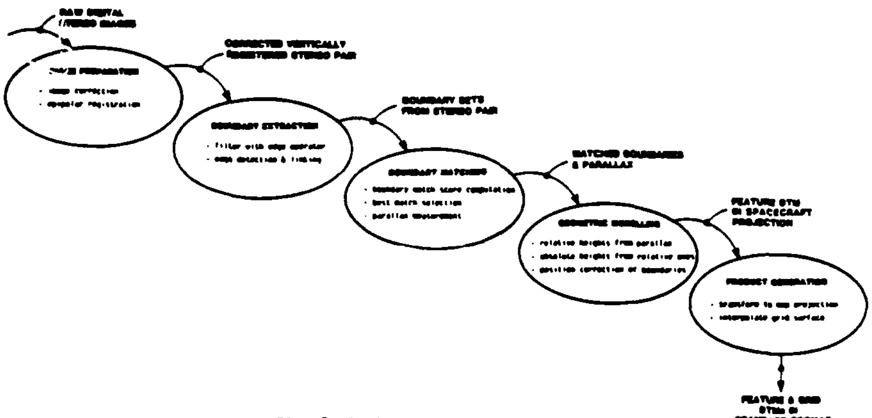


Fig. 2. Basic system structure

Establishing this vertical epipolar registration is complicated by a relative rotation between the images. This rotation, which is due to changes in the satellite's relative attitude at the two image acquisition points, is measured from previously marked ground control points. It is small enougn (on the order of one degree) that large regions of the image have the same vertical registration. This allows approximate overall epipolar registration by varying the vertical alignment in each region.

After all of this preparation, the result is a geometrically correct vertically registered pair of images in spacecraft projection. Figures 9 and 10 show a Landsat example.

Boundary Extraction

The purpose of this phase is to extract something from the images for matching which is invariant to confounding radiometric effects. Such effects include changes in point of view and illumination. The invariant selected should correspond to places where real changes occur in the world - in other words, boundaries. Changes in the world might be elevation changes like ridges or changes in reflectance caused by roads or fields. The corresponding boundaries in the image are assumed to be characterized by abrupt changes in the intensity function.

Our boundary detector is based upon those of Marr and Hildreth (1980) and MacVicar-Whelan and Binford (1981). The boundaries are detected at a variety of levels of resolution, to provide hierarchical information to the matching process. The extracted boundaries are described by boundary position (to subpixel precision), shape, and contrast (the intensity to either side).

The first part of boundary extraction is an image processing step: application of an edge operator or filter to the image by convolution. In this system, the function convolved with the image is a difference of two Gaussians, one positive, one negative. This effectively acts as a smooth second derivative of the image intensity function, which converts edges in the initial image to zero-crossings in the filtered image. It also serves to smooth the image to varying levels of resolution, when the width parameter of the mask is altered.

Next, a second convolution-like traversal with templates is used to actually detect the boundaries in the filtered image. This traversal locates, links, and measures the shape of

the zero-crossings, and disregards boundaries below thresholds of structural and radiometric significance. Boundaries parallel to the scan lines are regarded as insignificant, for example, because they cannot be used to measure parallax.

Some results of the boundary extraction process are shown in Figures 4 through 6. These are boundaries extracted at three levels of resolution from the Gun Lake image in Fig. 3, described later. Figure 4 shows boundaries extracted at the coarsest level of resolution. Only the most obvious image features (e.g. the top of the ridge, the sides of the lake) are present. Figure 5 shows more detail, but image noise has larger significance. Finally, Fig. 6 shows some of the finest resolution boundaries at the upper end of the lake surrounding the dam. The fine detail and obvious structure of the dam and road in the image are echoed in the boundaries, even though the features are sometimes only a single pixel wide.

Fig. 3 Gun Lake simulated SPOT image

Fig. 4 Gun Lake boundaries at coarse resolution

Fig. 5 Gun Lake boundaries at medium resolution

Fig. 6 Detail of Gun Lake boundaries at finest resolution

From the boundary extraction phase, the result is a list of symbolically characterized linked boundaries at varying levels of resolution (from each of the left and right images).

Boundary Matching

The system next attempts to correctly determine, for a given boundary from the left image, which boundary from the right image corresponds to it (if any). The processing occurs in a hierarchic zoom-in fashion, from the coarsest level of resolution to the finest. At coarser levels of resolution, signal-to-noise is better, and there are fewer boundaries and much less ambiguity in matching. As a result, coarser resolution matches are more robust and are used to constrain later matches at finer levels of resolution.

Actually determining the best match of boundaries at each level of resolution requires the following. First, there must be a mechanism for comparing potential left image/right image boundary pairings. In this system, this is accomplished by computing a statistical similarity

score between the boundaries being compared. Once this is done, the best overall set of such match pairings must be selected. While it is theoretically possible to test all potential combinations of such pairings, the number of such combinations is prohibitively large. To constrain this search, we use a dynamic programming technique similar to that of Baker (1982) and Ohta and Kanade (1985). Dynamic programming (Bellman, 1957) is a well known technique for computing a global optimum (in this case, the best overall match) efficiently.

When the best match of boundaries at the finest level of resolution (no smoothing) is determined, stereo parallax is computed by subtraction of the coordinates of the matched boundaries in their respective images. The net result of the entire matching process is a set of matched boundaries, and the associated stereo parallax everywhere along them.

Geometric Modelling

It is now necessary to solve the geometry problem of converting parallax measured in the images to depth in the world. The basic base-to-height model provides a starting point (LaPrade, 1980; Simard and Krishna, 1983). In this model, depth is a function of the measured parallax, the height of the satellite above the geoid, and the base or separation between the ground tracks of the satellite. Values for the height and base are obtained from the models of the satellite's attitude and orbit derived during the image correction phase. The parallax must be measured to sub-pixel precision to obtain sufficient accuracy. In this system, the boundary positions (and thus parallax) are easily computed to sub-pixel precision.

As is, however, this basic model is inadequate even for small test cases like those presented later. As a very minimum, it is necessary to compensate for the relative rotation between the images mentioned earlier, or large height errors result. The curvature of the earth's surface should be modelled as well. Processing real full-size SPOT images requires even more modifications to the model. The worst problem is the side-looking nature of SPOT's stereo imaging (relative to vertical or nadir looking), which introduces non-linear perspective distortion into the images. Correctly modelling the effects of this distortion in stereo imaging is extremely difficult.

Heights determined so far are all relative to a floating datum plane (defined by zero parallax). The second step in post-match processing is therefore to convert these relative heights into absolute heights, i.e. heights relative to the standard reference for the earth's surface, the geoid. The incorporation of one or two points of ground truth, so the relative values can be "tied down" to the surface, accomplishes this.

The next step is to place the features (and their depths) in the correct position. This is similar to the process of orthophoto generation, in that it is desired to place the boundaries where they would be if viewed from directly above (rather than from the satellite imaging position). Once again the stereo imaging model and the parameters derived during image correction are used, this time to produce a correction for the position distortion.

Product Generation

Now we have an actual "Feature DTM" - correctly positioned depths everywhere in the scene there is interesting structure. To obtain a finished product this feature DTM, which is still in spacecraft dependent projection, is rotated and scaled into standard map format. This is accomplished with standard mathematical warping transformations. Because there is not yet a grid of data, it is not necessary to resample the data for this new projection.

Obtaining a gridded DTM product requires one final step. The dense surface must be reconstructed from the sparse depth information along the boundaries. This is essentially a task of interpolation between the boundaries. Such interpolation, in two dimensions with irregularly spaced samples, is in itself an extremely difficult problem. Gold, Charters, and Ramsden (1977) describes the method we adopted, for convenience and simplicity.

TEST RESULTS

Overall Match Results

Feature DTM. Because the grid DTM is derived directly from the feature DTM, the quality of the initially produced feature DTM is of primary concern. The matching process, due to its coarse-to-fine and non-local nature, is extremely robust and always generates a feature DTM which is correct in essence. The results of the basic feature match, as shown in the summary in Table 1, are within target accuracy goals.

Table 1 Summary of Test Results

	Test Scene		
	Gun Lake (Simulated SPOT)	Death Valley (Landsat 5)	Vancouver Island (Landsat 5)
Without error removal filter			
Number of matched feature points	10203	11106	12429
Feature DTM accuracy	9.9 m	90.5 m	144.6 m
Grid DTM accuracy	9.2 m	78.5 m	130.8 m
With error removal filter			
Number of matched feature points	9208	9269	7874
Feature DTM accuracy	7.9 m	61.8 m	74.7 m
Grid DTM accuracy	7.3 m	59.5 m	69.8 m

All accuracies measure the RMS difference in metres between the satellite derived DTM and the reference DTM, over the entire area of the derived DTM. To compute the feature DTM accuracy, feature position was rounded to the nearest pixel.

As is to be expected however, the match is not perfect. Fortunately, the few errors that are made are easy to detect, easy to remove, and relatively insignificant. These errors are incorrect matches, when the wrong boundaries or boundary fragments are put into correspondence. The result is errors which are few, randomly distributed, point-like in character, and almost always in extreme difference with the surrounding terrain (either above or below). In contrast, most correlation-based matchers tend to err by "getting lost" completely, after which they cannot continue without human assistance.

Because the errors are so easy to detect, we designed a simple adaptive filter to remove them in a post-processing phase. The filter computes local statistics about terrain heights along boundaries, and throws out points which are in extreme difference. The effect of this filter can be best seen in Table 1. While only a fraction of the points are removed, almost all the actual match errors are taken out. This post-processing in no way constitutes smoothing of the terrain surface; all the detail is still present. It is simply the automatic removal of the more obvious random errors.

Grid DTM. Currently, the grid DTM is generated directly from the feature DTM by interpolation. As a result, the grid DTM tends to be most accurate at discontinuities and narrow features such as river edges, usually the most important locations for height, and usually where most correlation-based matchers are weakest. On the other hand, the interpolator we are currently using, based on triangular surface patches, tends to produce somewhat discontinuous and faceted surfaces between the boundaries.

We plan to use constrained area matching (such as correlation) between matched boundaries (Baker and Binford, 1981) to augment the results of the boundary matching, which should improve the quality of the output grid DTM. Use of any of the smoother interpolation algorithms would also improve the quality of the results.

Test Scenes

SPOT simulation imagery. Figure 3 shows the left image from the simulated SPOT stereo pair. The imagery for this test case is simulated in two senses (Simard, 1981). First of all, the actual imagery was not obtained by the SPOT satellite, but by an airborne multispectral linear array scanner. The spatial and radiometric characteristics of SPOT imagery (10 metre pixels from the panchromatic sensor) were then simulated from this input image. Secondly, the right image of the pair was synthesized from the left and a digitized contour map of the area; no real stereo imagery was acquired. The base-to-height ratio in the synthesized stereo pair is approximately 50 percent. The image itself shows a scene near Gun Lake British Columbia, Canada. Over the 3 km by 6 km scene, the terrain ranges from 600 to 1300 metres, including an abrupt height change at a dam.

Figure 7 shows the DTM generated automatically from the stereo pair, by matching and interpolating the boundaries show in Figures 4 through 6. (All DTMs are presented as images in this paper, with brightness corresponding to elevation; the higher the brighter. Each DTM has maximum contrast stretch for its terrain height range). Figure 8, for comparison with the stereo derived DTM in Fig. 7, is a reference DTM generated from digitized contour maps. (The smaller size of the stereo derived DTM is due to the size of the edge extraction filter. The jagged edges are the outermost matched boundaries).

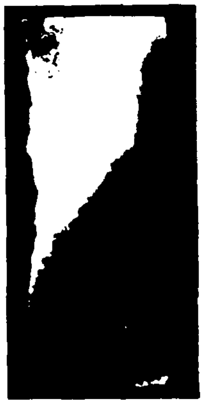


Fig. 7 Gun Lake stereo derived DTM

Fig. 8 Gun Lake reference DTM

The RMS error measured between the stereo derived DTM in Fig. 7 and the reference DTM in Fig. 8 is 7.35 metres. This is an unrealistically low error, no doubt due to the exact radiometric correspondence between the left image and the right image, which was itself synthesized by geometrically but not radiometrically altering the left image.

Death Valley imagery. Figures 9 and 10 show an actual satellite stereo pair. (These figures, as well as figures 17 and 18 can be viewed in three dimensions through a stereoscopic viewer). The images are of the Death Valley region of California in the United States and were acquired by Landsat 5 on June 30 and June 23, 1984, respectively. At this latitude, Landsat 5 images have about 25 percent overlap providing stereo coverage, from which the two 512 by 512 pixel chips were extracted. Both images are from band 5 of the Thematic Mapper (TM) sensor, with a pixel ground size of 30 m by 30 m. The base-to-height ratio for Landsat stereo pairs is about 20 percent. The terrain, including Telescope Peak at 3367 metres and nearby Panamint Valley at 500 metres, is extraordinarily rugged.

Figure 11 shows the grid DTM generated from the Death Valley stereo pair. Figure 12 is the reference DTM for comparison, generated from 1:250,000 scale contour maps by the United States Geological Survey (USGS). The resolution of this reference DTM is low; the pixel size (90 by 75 metres) necessitated digitally zooming both Figures 11 and 12 by a factor 2 for the pictures. The difference between the two DTMs is presented visually in Fig. 13 with the contrast stretched to make the errors visible. Against the grey of zero error, the low magnitude and random character of the overall error is apparent, and some triangular artifacts of the interpolation are visible.

The RMS difference between the satellite derived DTM and the reference is less than 60 metres over the entire area of the satellite derived DTM; in most places it is less. This is particularly impressive when the error in the reference DTM (up to 60 metres) is taken into account. Profiles taken vertically (column 364) through both DTMs can be compared in Fig. 14. It is easy to see that the satellite derived DTM has the correct terrain structure. Furthermore, much of the difference between the two is clearly due to the smoothing which has been done to the USGS DTM; considerable terrain detail in the image and satellite derived DTM is not present in the 1:250,000 scale reference. In some respects, the satellite derived DTM is actually better than a DTM derived from 1:250,000 scale maps. The satellite derived product and reference DTM are shown once again in Fig. 15 and Fig. 16 respectively, in a format similar to contour maps. The contours are at absolute intervals of 200 metres. Here again, the smoothing of the USGS DTM is obvious.

Vancouver Island imagery. Figures 17 and 18 show the stereo pair for the third test case, on the northern side of Vancouver Island near Kelsey Bay, Canada. The left image was acquired on July 15, 1984, while the right was acquired on July 24, 1984. Both images are TM band 4 with 30 metre pixels, 512 by 512 in size. In this case, the terrain is less severe (ranging from sea level to 1750 metres) and man-made features such as logged areas are present in the images.

The satellite derived DTM and the reference are shown in Figures 19 and 20. The reference DTM (which covers only part of the scene in this case) has 25 metre pixel resolution, which is even better than that of the source images. This results in a degradation in accuracy when

the satellite derived DTM is registered to the reference. Even at this resolution, however, the satellite derived DTM is correct to within 69.8 metres RMS overall. Inaccuracies due to the interpolation process, which are more apparent in the DTM at this level of resolution, can be easily removed by smoothing the DTM or using a smoother interpolation.

CONCLUSIONS

The results presented above indicate quite clearly that it is possible to automatically generate digital terrain models from satellite imagery. Furthermore, the accuracy of the DTMs which can be generated is sufficient to ensure their usefulness for tasks such as automatic terrain mapping.

The success of a stereo matching system rests almost entirely upon the quality of the matching process. For the system to be successful, it must be robust enough to handle a wide variety of input imagery, and it must be able to match the entire scene with minimal match errors and minimal human assistance. We have built and tested a system which meets these goals admirably, as demonstrated by its ability to derive DTMs of reasonable accuracy from a variety of real imagery. The critical strategies adopted in the design - the use of non-local structure in the image with the boundaries, and the hierarchic coarse-to-fine matching - were a necessity in obtaining a sufficient degree of robustness in the match process.

ACKNOWLEDGEMENTS

The work of the first author was supported in part by U.S. National Science Foundation Coordinated Experimental Research Grant DCR-8320136 to the Department of Computer Science at the University of Rochester, Rochester, New York (his current address). The majority of work done on the project was supported by an IRAP grant from the Canadian National Research Council to MacDonald Dettwiler (MDA). R. Simard of the Canadian Centre for Remote Sensing graciously provided the SPOT simulated imagery and Gun Lake DTM. The Canadian Centre for Remote Sensing supplied the Death Valley Landsat data and provided the Vancouver Island data through the B.C. Ministry of Environment. The B.C. Ministry of Environment also provided the DTM for that scene. Processing of the Landsat images on the MDA GICS system by Rick Jeffrey and Nancy Minielli is gratefully acknowledged. All the pictures in this paper were imaged on an MDA Color FIRE 240 film recorder. Thanks also to the GICS analysts, and the MDA support staff who assisted in the preparation of this paper.

REFERENCES

Baker, H.H. (1982). Depth from Edge and Intensity Based Stereo. Stanford University AIM 347, Stanford California.

Baker, H.H., and T.O. Binford (1981). Depth from Edge and Intensity Based Stereo. In Proc. 7th Int. Joint Conf. on Artificial Intelligence. pp. 631-636.

Barnard, S.T., and M.A. Fischler (1982). Computational Stereo. ACM Computing Surveys, 14, 553-572.

Bellman, R. (1957). Dynamic Programming. Princeton University Press, Princeton, New Jersey.

Chevrel, M., Courtois, M., and G. Weill (1981). *The SPOT Satellite Remote Sensing Mission.* Photogram. Eng. and Remote Sensing, 47, 1163-1171.

Friedmann, D.E. (1981). Two-Dimensional Resampling of Line Scan Imagery by One-Dimensional Processing. Photogram. Eng. and Remote Sensing, 47, 1459-1467.

Friedmann, D.E., Friedel, J.P., Magnussen, K.L., Kwok, R. and S. Richardson (1983). Multiple Precision Rectification of Spaceborne Imagery with Very Few Ground Control Points. Photogram. Eng. and Remote Sensing, 49, 1657-1667.

Gold, G.M., Charters, T.D., and J. Ramsden (1977). Automated Contour Mapping Using Triangular Element Data Structures and An Interpolant Over Each Irregular Triangular Domain. ACM Computer Graphics, 11, 170-175.

LaPrade, G.L. (1980). Stereoscopy. In C.C. Slama (Ed.), Manual of Photogrammetry, 4th ed. American Society of Photogrammetry, Falls Church, Virginia. pp. 519-544.

Marr, D., and E. Hildreth (1980). Theory of Edge Detection. Proc. R. Soc. B, 207, 187-217.

Marr, D., and T. Poggio (1979). A Computational Theory of Human Stereo Vision. Proc R. Soc. B, 204, 301-328.

MacVicar-Whelan, P.J., and T.O. Binford (1981). Line Finding with Subpixel Precision. Proc. Soc. Photo-Opt. Inst. Eng., 281, 211-216.

Ohta, Y. and T. Kanade (1985). Stereo by Intra- and Inter-Scanline Search Using Dynamic Programming. IEEE Trans. Patt. Anal. and Mach. Int., PAMI-7, 133-154.

Simard, R. (1981). Results of Stereoscopic Image Simulations for the SPOT HRV Carried Out at the Gun Lake Site in British Columbia. In Proc. 7th Canadian Symposium on Remote Sensing, Winnipeg, Canada. pp. 541-551.

Simard, R. and V.G. Krishna (1983). A Successful Approach in Three-Dimensional Perception of Stereo Landsat-MSS Images Over Cordilleran Relief. In Proc. 9th Int Symposium Mach. Proc. of Remotely Sensed Data. Purdue University, Indiana. pp. 31-40.

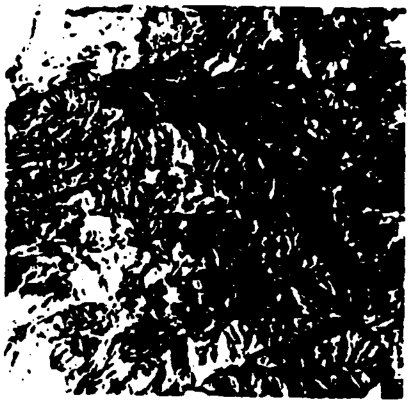


Fig. 9 Death Valley stereo pair left

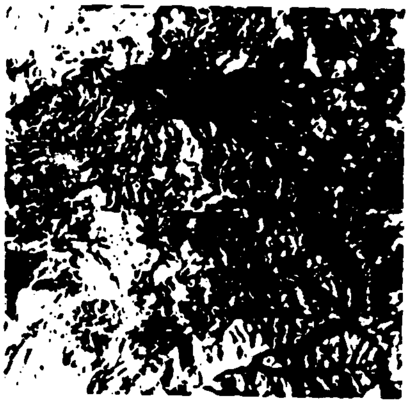


Fig. 10 Death Valley stereo pair right

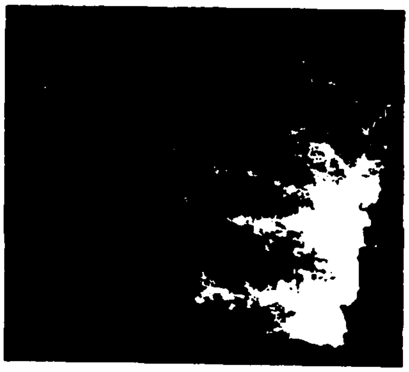


Fig. 11 DTM derived from Death Valley satellite images

Fig. 12 Reference DTM for Death Valley

Fig. 13 Difference between satellite derived DTM and reference DTM for Death Valley

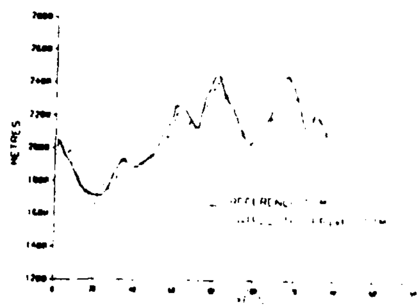


Fig. 14 Vertical profile plot through Death Valley DTMs

Fig. 15 Contoured DTM derived from Death Valley satellite images

Fig. 16 Contoured reference DTM for Death Valley

Fig. 17 Vancouver Island stereo pair left

Fig. 18 Vancouver Island stereo pair right

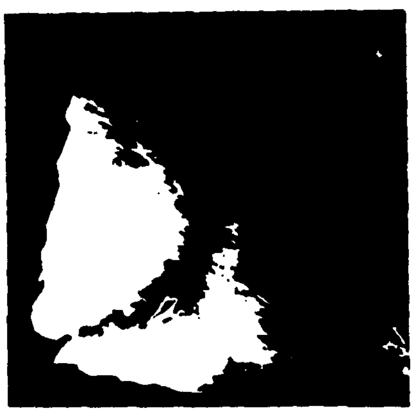


Fig. 19 DTM derived from Vancouver Island satellite images

Fig. 20 Reference DTM for Vancouver Island

Appendix B-5

Subgraph Isomorphism on the BBN Butterfly Multiprocessor*

John Costanso   Lawrence Crowl   Laura Sanchis
Mandayam Srinivas

Department of Computer Science
University of Rochester
Rochester, NY 14627

October 7, 1986

Abstract

This report describes an algorithm for finding subgraph isomorphisms for a restricted class of graphs and a parallel implementation of the algorithm on the BBN Butterfly Multiprocessor. This effort was part of a larger project to assess the suitability of the Butterfly architecture for a variety of machine vision tasks. Our algorithm searches a tree in which each node represents a partial assignment of vertices in the smaller graph to vertices in the larger graph. The algorithm prunes the search tree using properties of the two graphs as constrained by the partial mapping. These properties are vertex connectivity, distance between vertices, and the local topology of vertex clusters. By carefully balancing the computational load and the contention for shared resources, our algorithm achieves almost linear speedup in the processing rate of search tree nodes. However, the speedup of isomorphism detection rate is poor when looking for few isomorphisms, and good only when looking for many isomorphisms. We present an analysis of why we believe this property is intrinsic to algorithms that parallelize the tree search without parallelizing the node computations. We also discuss the effectiveness of the Butterfly architecture and programming environment in implementing such parallel algorithms.

*This work was supported in part by the Defense Advanced Research Projects Agency U.S. Army Topographic Labs under grant number DACA76-85-C-0001, in part by the National Science Foundation under grant number DCR-8320136 and in part by an AT&T Foundation Fellowship.

1 Introduction

The report describes the results of a project to implement a parallel algorithm for subgraph isomorphism on the BBN Butterfly multiprocessor. This project is part of a larger effort to assess the suitability of the Butterfly for implementing parallel algorithms for machine vision [Br86]. Our results indicate that the Butterfly is suitable for implementing an efficient algorithm for our problem.

1.1 Problem Refinement

The original problem statement, under the title of "Graph Matching", is as follows:

> The input is a graph G having 100 vertices, each joined by an edge to 10 other vertices selected at random, and another graph H having 30 vertices, each joined by an edge to 3 other vertices selected at random. The output is a list of occurrences of (an isomorphic image of) H as a subgraph of G. As a variation of this task, suppose the vertices (and edges) of G and H have real-valued labels in some bounded range; then the output is that occurrence (if any) of H as a subgraph of G for which the sum of absolute differences between corresponding pairs of labels is minimum.

The above problem statement includes two problems. The first problem requires us to enumerate every isomorphism. Because of the regularity of G and H and because vertices in G have much larger degree than vertices in H, there are likely to be a very large number of isomorphisms. We suspect that the average number of isomorphisms is exponential in the size of H. Thus, any program running on a computer such as the Butterfly would require prohibitively large amounts of computation, even on the average. (Note that even the problem of determining the existence of an isomorphism is NP-complete, and is commonly referred to as the Subgraph Isomorphism Problem [GJ79]). Since we are mainly interested in assessing the suitability of the Butterfly architecture for this problem under a very tight development time constraint, we decided to focus on the average time required to find a small, fixed number of isomorphisms, (*e.g.*, 10 or 100) rather than the average time to enumerate all of them.

The second problem is a generalization of Subgraph Isomorphism to graphs with edge and vertex costs. Here each isomorphism is assigned a cost according to some formula, and it is required to find the minimum cost isomorphism. Again, because of development time constraints, we do not address this problem in this report.

Finally, a word about terminology. The word *matching* has a specific meaning in graph theory, namely a set of vertex-disjoint edges in a graph. Graph Matching would thus suggest one of the well-known problems that involve finding such matchings in graphs. We therefore prefer to use the standard terminology, and will hereafter refer to our problem as Subgraph Isomorphism.

1.2 Approach

Our parallel algorithm is based on a modification of Ullmann's sequential tree-search algorithm for subgraph isomorphism [Ul76]. Ullmann's method generates a search tree where

each node in the tree represents a possible partial isomorphism. Let $v_1, \ldots, v_n$ be the vertices of H, ordered so as to correspond to the depth in the search tree. A node at depth k contains a single mapping for vertices $v_1, \ldots, v_k$ and a set of possible mappings for vertices $v_{k+1}, \ldots, v_n$ At the leaf nodes of the tree, all vertices, $v_1, \ldots, v_n$, have a single mapping and the node represents an isomorphism.

Ullmann's algorithm prunes the search tree by eliminating mappings that are infeasible because they violate connectivity requirements. We view this procedure as the application of a *connectivity filter*. The regularity of the graphs for our problem is such that the connectivity filter will be less effective than in the general case. Because of this, we have attempted to improve the pruning efficiency of Ullmann's algorithm by precomputing a good vertex traversal order and by adding two new filters, the *distance* and *configuration* filters.

There are basically two aspects of the sequential algorithm that can be parallelized: the computation at each node and the exploration of the search tree. We have chosen to focus on the latter aspect for two reasons. Parallelizing the exploration of the search tree has the advantage of requiring minimal communication between processors, which implies low synchronization and contention costs. We also believe that it is the more interesting aspect, since the node computations tend to represent matrix computations, and parallelizing such computations on the Butterfly is better understood.

The basic algorithm is to generate the graph, perform the precomputations, and spawn off a number of processes to search the tree. The precomputations include ordering the vertices to increase search effectiveness, and precomputing constant information for the distance and configuration filters. Each process searching the tree contains a loop in which it obtains a search tree node from the shared stack, applies the filters to the node, and places the feasible children of the node, if any, on the stack. The loop continues until either the appropriate number of isomorphisms has been found, or there are no more nodes left to process. For a single processor, this algorithm is equivalent to a depth first search of the tree.

1.3 Results

Our algorithm was successfully implemented on the Butterfly. Experiments were conducted for finding between 1 and 100 isomorphisms, using between 1 and 96 processors. Figures 1-4 summarise the experimental findings.

The program exhibits almost linear speedup in the node processing rate. However, the solution rate (*i.e.*, the number of isomorphisms detected per second) shows good speedup only when the program is looking for 100 isomorphisms. The solution rate speedup is poor when looking for fewer isomorphisms, a phenomenon that we believe is intrinsic to algorithms that parallelize the tree search without parallelizing the node computation. Going in the other direction, we have evidence to suggest that the solution rate speedup will be almost linear if we are looking for a very large number of isomorphisms. However, the actual solution time would be prohibitive in that case.

Our effort indicates that the Butterfly architecture provides an effective balance of features for implementing parallel algorithms of the kind likely to be used for problems similar to ours. However, architectural suitability alone does not guarantee effective utilization

of a parallel processor. The machine must also be equipped with a workable programming environment. We found that the Butterfly's programming environment, evidently reflecting the state of the art in parallel programming environments, is currently inadequate for rapid and reliable program development.

2 Algorithm Details

This section describes the details of our algorithm. We examine random graph generation, search vertex traversal order determination, and search tree node filtering.

2.1 Graph Generation

Generating a graph according to the given specifications is an interesting problem in itself. Consider an incremental algorithm to generate the graph H, which has 30 vertices each connected at random to 3 others. Thus H is a 3-regular graph. The 3 neighbors of some vertex v must be chosen in such a way that the triple that is chosen has the same probability as any other feasible vertex triple. (A triple is feasible if it does not violate the degree constraint for any vertex in the triple). Incrementally, we must ensure that the next vertex added to the triple is chosen at random from the remaining feasible neighbors.

It is not hard to see that such a procedure might not always yield a graph H, since we might run out of feasible neighbors prematurely, before all vertices have received their full complement of 3 edges each. We illustrate this phenomenon with a small example. Let K_4 be the complete graph on 4 vertices and $K_{3,3}$ be the complete bipartite graph with 3 vertices in each partition. Both K_4 and $K_{3,3}$ are 3-regular. Suppose it is required to generate a 3-regular graph with 6 vertices. The procedure described above might incrementally generate K_4. At that point, 4 vertices have degree 3 and 2 have degree 0. The only remaining feasible edge will connect the two zero-degree nodes, which will then each have degree 1. Since no further edges can be added, the procedure fails.

If we relax the requirement that the neighbors of a vertex must be chosen at random, it is simple to generate a 3-regular graph on n vertices, for n even. (Exercise for the reader: prove that for odd n, no 3-regular graph exists). We simply have $\lfloor n/4 \rfloor - 1$ copies of K_4 and one copy of either K_4 or $K_{3,3}$. However, it is required to choose neighbors at random. We therefore strengthen our definition of neighbor feasibility.

A vertex w is a feasible neighbor of v if adding the edge (v, w) does not violate the degree constraint for either v or w, and, furthermore, after adding the edge (v, w), the residual degree sequence is feasible. The *residual degree* d_i of a vertex i is the number of edges it needs to bring its degree up to 3. The *residual degree sequence* of a graph is formed by arranging the residual vertex degrees in non-increasing order. It can be shown [BM76] that the residual degree sequence is *feasible* if, for $1 \leq k \leq n$, $\sum_{i=1}^{k} d_i \leq k(k-1)+\sum_{i=k+1}^{n} \min\{k, d_i\}$. If a residual degree sequence is feasible, then we know that the construction can be completed with the edge (v, w) included.

Following the approach in [Ti79], our graph generator uses the neighbor feasibility test on-line. Every time a neighbor is examined, the degree constraint and the residual degree criterion are checked. The neighbor is deemed feasible only if it passes the test. Three

feasible neighbors are then chosen at random for each vertex. The 10-regular 100-node graph G is similarly generated.

2.2 Vertex Ordering

The search procedure assumes a vertex traversal order. In this section we show how to choose this order to our advantage. Consider the situation in the algorithm after vertices $v_1, \ldots, v_k$ have been mapped. An unmapped vertex that is connected to a larger number of mapped vertices will have more restrictions on possible mappings. Thus, our goal is to choose the next vertex to visit in the search as the one that is maximally connected to vertices already mapped. We believe that this procedure will tend to cause pruning earlier rather than later, by applying restrictions to the edge mappings as soon as possible. This will make the search tree sparser and reduce the search space.

We compute a vertex ordering as follows. For each vertex we record a label, representing its position in the eventual ordering, and its residual degree. Initially every vertex is unlabeled and has residual degree 3. At each step $i, 1 \leq i \leq n$, we choose an unlabeled vertex of minimum residual degree, label it with i, and decrement the residual degree of its unlabeled neighbors. The procedure thus tends to choose vertices highly connected to the vertices already labeled.

2.3 Node Filters

We have implemented three filters, a connectivity filter modified from Ullmann's original filter, a distance filter based on the distances between vertices in the two graphs, and a configuration filter based on the possible configurations a vertex and its three neighbors can assume in the H graph. Note that the distance and configuration filters, when combined with use of the connectivity filter to restore consistency after 1's have been eliminated, are a generalization of, and hence more powerful than, the connectivity filter alone.

Ullmann's Approach

In Ullmann's implementation, each node of the search tree is associated with an m by n binary matrix M where m is the order of H and n is the order of G. The (i, j) entry of M is set to 1 if we are still considering the possibility that vertex v_i of H may be mapped to vertex w_j of G. At depth d in the search tree, M contains a single 1 in each of its first d rows. To generate the children of a node at depth d, we clear all but one of the 1's in row $(d+1)$ of M and zero out the rest of the column occupied by the remaining 1. So the branching factor is the number of 1's in row $d+1$. At the leaf nodes, exactly one 1 is set in each row of M and each column has at most one 1 set. To create the root matrix M we set the (i, j) entry of M to 1 unless we have reason to believe that vertex v_i of H cannot be mapped to vertex w_j of G.

Ullmann describes a refinement procedure which at each node attempts to get rid of 1's by checking connectivity requirements. If this removal results in a row of M losing all its 1's, the node can be pruned from the search tree. The connectivity check is as follows. For each 1 in M, say at position (i, j), the following check must be made. Let v_x be any neighbor of v_i in H. Then there must be some 1 in v_x's row in M corresponding to some

neighbor of w_j in G. That is, there must be some y such that w_y is connected to w_j in G and entry (x, y) of M is set to 1. If no such y exists, then the mapping from v_i to w_j is impossible and we may set entry (i, j) of M to zero. Such a check must be done for every neighbor v_x of v_i. After all 1's in M have been checked in this way, the process must be repeated if any 1's were eliminated during the pass. Checking connectivity at a leaf node determines whether or not we have found an isomorphism.

Our Modifications

Ullmann's connectivity filter, or refinement procedure described above, can be made more efficient by making the following 2 observations:

1) It is not necessary at each node to check each 1 in the M matrix for connectivity consistency. In fact, it is only necessary to check at each stage the neighbors of those H vertices whose rows in the M matrix have lost 1's since the last connectivity check. Of course, since this check may result in some 1's being eliminated, it may recursively cause more checks. This implementation of the connectivity filter is described in more detail below.

2) At depth d, the vertices in rows 1 through d need not be checked even under the circumstances described in (1). This is because these vertices have only one 1 set in their rows in M, and by previous connectivity checks all 1's in their neighbors' rows must correspond in G to neighbors of the vertex indicated by the single 1. In particular, this means that at leaf nodes of the search tree no checking needs to be done at all, and whenever we reach such a leaf node we are guaranteed to have found an isomorphism.

We now describe the procedures used to eliminate 1's at the beginning and throughout the search.

Connectivity Filter

This filter is invoked by giving it a list of vertices in H whose rows in M have recently lost 1's. This can happen, for instance, when new nodes are generated by branching in the search tree, or when other filters, described below, have been applied, resulting in the elimination of 1's.

The filter maintains a stack containing the numbers of the vertices whose neighbors must still be checked. This stack is initialized with the list of vertices passed in to the procedure. The algorithm proceeds by repeatedly taking vertices off the stack and processing them; this may result in more vertices being pushed onto the stack. The procedure ends when the stack is empty.

The following is done for each vertex v_i popped off the stack. If a 1 is in position (i, j) of M, then the jth row of the incidence matrix of G is retrieved. This vector will have a 1 corresponding to each neighbor of w_j in G. All such vectors are retrieved for each j such that (i, j) is 1 in M, and these vectors are ORed together. The resulting vector is ANDed with each row in M corresponding to a neighbor of v_i whose vertex number is not less than d, the current depth. For each such neighbor of v_i, the result of the AND operation is installed as the new row in M for this vertex. If the row for the vertex in M has been zeroed out, the corresponding node is pruned from the search tree. If one or more 1's have been eliminated from the row, the vertex is pushed onto the stack if it is not already there.

Distance Filter

We define the distance between two vertices in a graph to be the smallest number of edges that must be crossed in order to get from one vertex to the other. Clearly, the distance between the isomorphic images in G of vertices u and v must be less than or equal to the distance between u and v in H, since G has an image for every edge in H and potentially contains some shortcuts between the images of u and v.

This observation forms the basis of the distance filter, which we describe in this section. The distance filter requires knowledge of the distance between every pair of vertices in both G and H. We precompute these in two matrices by repeated breadth-first search starting at each vertex in G and H.

The distance filter may be applied every time a new node has been generated. Suppose a node has been generated at depth i by zeroing out all 1's except one in row i of M. Assume the remaining 1 is in column j. For each 1 in each row of M after row i, the following is done. Let the 1 be in position (l, m) of M $(l > i)$. That is, the 1 signifies that vertex v_l in H may be mapped to vertex w_m of G. For this to be so the distance in G from w_j to w_m must be less than or equal to the distance in H from v_i to v_l. If this check fails, then the 1 at (l, m) is zeroed out.

If the distance filter application results in elimination of 1's in one or more rows of M, the connectivity filter is invoked to restore connectivity consistency.

Configuration Filter

The configuration filter is based on the fact that because every vertex in the H graph has degree 3, one can enumerate the 4 types of configurations that a vertex v and its 3 neighbors can be in.

Type 0 configuration occurs when none of the neighbors of v are connected to each other by an edge. Type 1 configuration occurs when exactly two of the neighbors of v are connected to each other. Type 2 configuration occurs when exactly two pairs of the neighbors of v are connected to each other. Type 3 configuration occurs when all three neighbors of v are connected to each other, forming an isolated clique of size 4. Each vertex in H can be determined to be in exactly one of these 4 types.

In the graph G, each vertex has degree 10. If a vertex v in H is mapped to a vertex w in G, then the 3 neighbors of v in H must be mapped to three neighbors of w which together with w form a configuration having at least the connectivity of v's configuration. In other words, if v has configuration type k, v and its three neighbors must be mapped to a vertex w and 3 of w's neighbors forming configuration type l where $l \geq k$. Each vertex w in G has 120 3-tuples of neighbors and a configuration type corresponding to each of these 120 3-tuples.

Application of the configuration information requires knowledge of the configuration type of each vertex in H and of the types associated with each vertex in G and each of its 3-tuples of neighbors. This information is computed and stored at the beginning of the algorithm.

One application of the configuration information takes place when the matrix M is initialized at the root node of the search tree. At this point each feasible mapping of a vertex v_i to a vertex w_j in G is checked, to make sure that for some 3-tuple of neighbors of

w_j, w_j has configuration type at least as large as that of v_i. If this is not the case, the (i, j) entry of M is zeroed out.

In addition, a configuration filter may be applied whenever a new node is generated in the search tree. Suppose a new node has been created at depth i. This has been done by zeroing out all 1's in row i of M except one, say in column j. Thus for all isomorphisms that may be found in this branch of the search tree vertex v_i of H is mapped to vertex w_j of G. We want to make sure that the neighbors of v_i are all mapped to neighbors of w_j which form the correct type of configuration with w_j. In order to do this the filter takes each 3-tuple of neighbors of w_j whose configuration type is at least as large as v_i's configuration type, and records the fact that each of v_i's neighbors may be mapped to any one of the vertices in the 3-tuple being considered. After this is done for each applicable 3-tuple of neighbors of w_j, the 1's corresponding to mappings for the neighbors of v_i which were not recorded in the above procedure are zeroed out in the matrix M. Since this process can result in elimination of 1's in the rows of M corresponding to v_i's neighbors, the connectivity filter should be then called to restore connectivity consistency.

3 Implementation

The program is implemented on the Butterfly using the C programming language and calling the Chrysalis operating system [BB85a] directly. Programming with Chrysalis is appropriate when there is need for shared data but not for a shared address space, and when the granularity of processes is high. The algorithm chosen has these characteristics.

The implementation has a main process to setup the computation, and several server processes to perform the tree search. The main process *initializes* a *shared stack*, consisting of tree nodes which have yet to be searched, to the set of possible tree nodes at depth one. The server processes extract a node from the shared stack, filter it, and place any feasible children back on the shared stack.

The main process: sets up the shared memory environment through which all processes will communicate; generates the random graph and first search tree node; precomputes the vertex order, static distance filter information and static configuration information; spawns the server processes; signals the processes to start searching; and waits for the processes to finish.

Each server process: connects to the shared memory environment; waits for the start signal; copies the constant problem and precomputation information into local memory; loops processing search tree nodes until the appropriate number of isomorphisms have been found; and signals the main process upon termination. To process a search tree node, the server process: retrieves a tree node from the shared stack; applies the filters to the node; and places any feasible child nodes back on the shared stack.

3.1 Sequential Operations

Not all elements of the program could be parallelized in the short time available to our project. As a result, some parts were implemented sequentially. This section discusses the reasons each was left sequential.

Graph Generation

We did not consider the random graph generation as part of the problem statement, so no effort was made to parallelise it.

Precomputation

The precomputation is graph dependent, and therefore is necessarily part of the problem statement. We did not parallelise the precomputation since it took only about 12 seconds on a single Butterfly node, which represents a small fraction of the total work. In addition, the precomputation is a relatively straightforward matrix computation, and parallelizing such computations on the Butterfly is relatively well understood. Therefore we did not feel that the coding effort would be justified. This precomputation time is not counted in the solution times cited later.

Spawning Processes

The server processes are sequentially started by the main process. We made no attempt to parallelize process startup because we feel that, in the context of vision, the isomorphism search is likely to be repeatedly called on a stream of problems. In such an environment, the processes would be started and left started, and the relative cost of process startup would become insignificant. Because of the time to start processes, a process spawned early could conceivably do a substantial amount of work on the search while other processes were still being started. This would adversely affect the timing results, so all processes start computation only when receiving a broadcasted "start" signal.

3.2 Parallel Operations

This section describes those aspects of our implementation that we have parallelized. These parallelizations strive to keep processors as independent as possible while keeping them as busy as possible.

Copying Precomputation Results

The precomputation results are placed in a central location by the main process. While the server processes could use this central location to access the appropriate data, such accesses would be subject to contention with the other server processes attempting to access the information. Such contention can result in a substantial performance degradation. Therefore, we have each server process copy the precomputation results to the local processor for the duration of the search. The access costs to the data are minimal with the local copy.

Tree Search

An obvious approach for parallelizing a tree search with such a large branching factor is to assign a processor to each top level subtree and have each processor then execute a sequential algorithm on the smaller problem. This approach has the advantage that there is almost no communication between processors, and hence they will not contend for information

resources. There is, however, a disadvantage in that if a processor finishes early (because it eliminates a subtree), there is no work left for the processor and it becomes ineffective

The first approach implemented had a central stack for all tree nodes. This eliminated the ineffectiveness of a processor finishing early. Unfortunately, the stack tended to grow very large. Due to the parallel, random, nature of stack access, the algorithm no longer has the space preserving properties of a depth first search. Indeed, the stack may grow as quickly as it would for a breadth first search, which would store a prohibitively large number of tree nodes for this problem. In addition, processors were always contending for access to the shared stack.

The second approach implemented used the shared stack only for search tree nodes at depth one and two. Below that, a local stack was used for searching a subtree in a strictly depth first manner. This reduced overall contention for the shared stack while still leaving plenty of work for a processor which finished a subtree early.

Shared Stack

The shared stack is implemented as a data structure in shared memory. Each server process is responsible for consistently updating the stack. Because server processes will start execution of the stack at arbitrary times relative to each other, these operations must insure that they do not interfere with each other. We solve this problem by having the server processes lock the stack using an atomic test and set operation when performing the critical data structure updates. When a server process finds the stack locked, the process busy waits until it finds the stack unlocked. Locking the stack produces contention for the stack. This contention in turn reduces the amount of effective parallelism that can be obtained from the program. Therefore, the program should seek to minimize the amount of time a process locks the stack relative to the amount of time processing away from the stack.

4 Analysis of Results

Our algorithm was successfully implemented on the Butterfly. Data was collected for s = 1,5,10,20 or 100, where s is the number of isomorphisms being sought, with p = 1,2,4,8,16,32,64 or 96 processors. For comparison purposes, each Butterfly node is about 50% slower than a Sun 2/50 workstation. Figures 1-4 summarize the experimental findings. Each data point represents between 6 and 10 trials. The majority of our experiments were run on a single, randomly generated instance of each of the graphs G and H. More robust results would be obtained if we were to run our program on different random graphs.

Our experiments indicate that using both the distance and the configuration filter is only marginally more useful than using either of these filters alone. In addition, we found that the distance filter took less time to run. Because this latter fact was observed too late in the experimentation, the configuration filter alone was used for all of our trials. However, the nature of the parallelization is such that using any combination of filters at each node should yield the same qualitative results in terms of speedup.

4.1 General Analysis

This section provides a general analysis of the parallel algorithm. Analysis specific to our implementation is discussed in the next section.

Node Processing Rate

Figure 1 shows the number of search tree nodes processed per second as a function of the number of processors. Our program achieves a near linear speedup in the node processing rate, indicating that our program is an effective parallelization of the tree search algorithm.

Solution Rate

Figure 2 shows the number of solutions generated per second as a function of the number of processors. This figure clearly shows that the program did not achieve linear speedup in problem solution time. One may observe, however, that the speedup gets better as larger numbers of isomorphisms are sought. This seems to confirm the conjecture that near linear speedup would be achieved by our program if we were to look for a sufficiently large number of isomorphisms. This conjecture is supported by the following analysis.

Suppose we are looking for one isomorphism. Consider a search tree with two subtrees. A sequential algorithm will explore one subtree first, and then, if a solution is not found, will explore the second subtree. In parallelizing this search for two processors, we would send one processor down each of the subtrees. Now, assume the solution is found in the first subtree. The first processor will find the solution while the second processor will not. However, the first processor will find the solution in the same amount of time as the sequential version. The second processor will contribute nothing to the problem solution time.

This argument can be extended to multiple processors, and multiple solutions. As different processors explore different branches of the search tree, some will find solutions and some may not. Whether or not a processor finds a solution depends on whether or not the processor can find a solution before the required number of solutions are found by other processors. The extent to which processors search without yielding solutions determines the amount of unprofitable work expended. If some of this work would not take place when using only one processor, then this work represents a limit to the speedup that can be achieved with parallelism.

Consider the case where we are searching for a few isomorphisms and the search tree contains a large number of isomorphisms distributed evenly among its branches. A sequential algorithm would only need to go down a few branches to find the required isomorphisms. The parallel implementation, however, would explore many branches at once, and would terminate when an appropriate (small) number of processors have found isomorphisms. In the meantime, the remaining processors would have expended a considerable amount of work. Thus parallelization in this case is only marginally productive.

Now consider the case where we are searching for a large number of isomorphisms and the search tree contains a large number of isomorphisms distributed evenly among its branches. Most processors will contribute solutions before the appropriate number of isomorphisms

have been found. Thus, processors will contribute a larger percentage of their total work towards the set of solutions, and the efficiency of the parallel algorithm will be higher.

Effective Processors

Figure 3 shows the number of effective processors as a function of the actual number of processors. This figure again points out the greater effectiveness of parallelization when a greater number of isomorphisms is being sought.

Solution Time

Figure 4 shows the time in seconds for finding a solution as a function of the number of solutions being sought. Note that the unit cost of finding a solution does not increase as more isomorphisms are sought, but instead seems to slightly decrease.

4.2 Implementation Analysis

A high degree of contention for a shared resource can severely degrade overall problem solution time. Much of the art of parallel programming consists of balancing the need for minimizing contention at shared resources with the necessity of providing enough shared information for the processors to work effectively.

The current program still has contention for the shared stack, especially at startup, which is probably why we achieve slightly less than linear speedup for the node processing rate. In addition, when looking for 1 isomorphism, the number of nodes evaluated per second actually decreases beyond $p = 64$. We suspect this may be because there is an insufficient amount of work to overcome the initial shared stack contention.

Given more time, we would revise our technique for assigning the initial tree nodes to processors. The first approach we would take at this point would be to arbitrarily assign the tree nodes at depth one to the processors as in the naive approach. However, instead of keeping an entire subtree to itself, each processor would place on the shared stack those tree nodes at depths two or three that it will not immediately process. This would restrict startup contention to that caused by placing children on the shared stack, which is significantly less expensive than retrieving children from the shared stack.

4.3 Other Parallelizations

This section discusses some approaches that we might take in parallelizing some other aspects of the computation, such as the precomputations and the node filtering.

The node filters and the precomputations are array based and may generally be parallelised with a "parallel for-loop". As such, they are more suitable for implementation under the Uniform System [BB85b]. Parallelizing these computations can be effective. However, the precomputation results and individual tree search nodes would become shared resources and contention for these resources could become significant.

Since the number of available processors is limited, an interesting question arises regarding the tradeoff between the two aspects of parallelization. In other words, what fraction of the processors should we devote to speeding up the node computations, and what fraction

should we devote to exploring more nodes. One advantage of parallelizing node computations is that it does not entail sending processors down unproductive subtrees.

The following subsections describe some of the ways in which these parallelizations could be performed.

Vertex Ordering

The current sequential vertex ordering algorithm chooses arbitrarily among vertices contained in the bin with the minimal residual degree. This algorithm may be parallelized by having several processors extract nodes from the appropriate bin. This approach requires care to ensure that the actions of one processor reducing a vertex's residual degree do not adversely affect the processing of another vertex.

Distance Precomputation

The distance matrix may be parallelized by assigning each processor to a distinct vertex in the two graphs and having the processor compute the distances to all other vertices. The parallelized distance computations are more appropriate for implementation under the Uniform System.

Configuration Precomputation

Each processor is assigned a vertex in each of the graphs. Each processor then independently computes the vertex's configurations.

Connectivity Filtering

The stack based connectivity filter would be parallelized in much the same manner as the stack based tree search. However, such parallelization would require careful thought to ensure that the algorithm would not be affected by one processor turning 1's into 0's while another processor examines that information.

Distance Filtering

The distance comparisons are independent and may be distributed among the different processors.

Configuration Filtering

Parallelization of the configuration filter may be achieved by distributing the work associated with each 3-tuple of the neighbors of the mapped G vertex among the different processors.

5 Conclusions

The purpose of this exercise has been to examine the suitability of the Butterfly computer for solving certain versions of the subgraph isomorphism problem involving fixed size regular graphs. Since our interest is mainly in studying the effectiveness of parallelization, we

decided to restrict ourselves to finding a small number of isomorphisms. We also chose to parallelize the exploration of the search tree since the effect of such a parallelization is interesting but relatively unexplored, both in theoretical and practical terms.

5.1 Performance

The program showed effective processor utilisation in terms of the gross amount of work done. The average number of search tree nodes processed per second increased almost linearly with the number of processors. This supports our claim that our program is an effective parallelization of our algorithm.

We also found that using more processors resulted in faster detection of isomorphisms, although the speedup is not linear in the number of processors. The speedup is reasonably good for $s = 100$, but is poor for smaller values of s. Our results suggest that the speedup will be almost linear if s is very large. We believe that this behavior is inherent in the nature of the approach to the problem, and independent of the architecture used to implement the approach.

We also found that in general, the unit cost of finding an isomorphism decreased slightly as the number of isomorphisms sought increased. We did not look for more than 100 isomorphisms, however, since those alone took the program more than an hour to find using 1 processor.

5.2 Butterfly Suitability

The Butterfly is suited to the types of parallelization techniques employed in our program. In general, we find the Butterfly architecture is an effective parallel architecture, but that the programming environment needs significant work. The weakness of the programming environment should be considered more of a statement on the state of the art in parallel programming rather than a statement on BBN or the Butterfly.

Architecture

There is a cost associated with ability to share information, regardless of whether or not information is actually shared. Unfortunately, architectures which seek to minimize this cost tend to raise the cost of sharing information. A balance must be struck between the cost of having access to shared information when it is not being used, and the cost of accessing shared information when it is used. We believe the Butterfly architecture strikes a good balance.

The Butterfly provides a number of processor nodes, each containing a processor and local memory, which are connected via a high-speed omega (butterfly, fft) network. The memory management hardware determines whether or not an address references the local memory, or the memory on another node. When all processors are accessing local memory, there is no contention for memory resources, and the cost of the ability to share information is limited to a simple test in the address translation. This model has significantly lower cost for local accesses than those models which provide a single global memory for all processors. In addition, it scales to larger numbers of processors far more easily.

When access to shared data is needed, processes can place memory on remote nodes into their address space. This allows access to shared memory, at the cost of a higher memory cycle time. While this remote access is more costly than those architectures with a single global memory, the cost to access shared information is lower than message based architectures. The use of shared memory allows concurrent access to data structures which avoids the process interaction and synchronisation costs that message based systems inherently force.

When transferring large amounts of data from one processor's memory to another's, message passing implementations may be more appropriate. This is because the extended cycle time of remote memory references and repeated instruction interpretation can become expensive. For such situations, the Butterfly architecture provides a hardware implemented block memory copy between processors. With this hardware, message based communication between processors can be efficiently implemented. In addition, with the shared memory capability discussed earlier, a message based communication does not require interrupting the destination processor.

In summary, the Butterfly architecture provides local memory for efficient access to information that is not shared, an interconnection network for efficient access to remote memories for small amounts of shared information, and block memory copy for efficient sharing of large amounts of information. Any single feature could doubtless be implemented at a lower cost, but it is the balance of these features which determines the effectiveness of a parallel architecture.

Programming Environment

We find the programming environment for the Butterfly is its weakest attribute. The Butterfly architecture allows efficient implementation of algorithms based on a variety of models of computation, unlike many machines where the model of computation is built into the machine. On the Butterfly, programmers are not locked into a model of computation, but may choose that which seems most natural to the problem at hand. Unfortunately, in adopting a model of computation, Butterfly programmers must choose a programming environment which supports that model. Currently, all programming environments support a single model of computation. These environments then have a severe, often restricting, impact on the algorithms chosen. In other words, the current programming environments do not reflect the flexibility of the architecture.

We chose to program using Chrysalis directly rather than using the Uniform System. Chrysalis is more suitable for the large processes communicating through relatively disjoint regions of shared memory. This characterises our shared stack tree search algorithm. The Uniform System is suitable for those instances where one thinks in terms of shared matrices and "parallel for loops". This characterizes the types of computations needed for parallelising the precomputations and tree node filters. Further efforts at parallelizing would probably involve re-coding the program under the Uniform System for the precomputations and node filters and then working the tree search in on top of that. This brings up the issue of mixing models within a single program. These issues are an active research area, and more work is needed in this area.

References

[BB85a] BBN Laboratories; *Chrysalis Programmer's Manual*; Version 2.2, June 1985

[BB85b] BBN Laboratories; *The Uniform System Approach to Programming the Butterfly Parallel Processor*; Version 1, October 1985

[BM76] J. A. Bondy and U. S. R. Murty; *Graph Theory and Applications*; North-Holland, 1976.

[Br86] C. Brown, R. Fowler, T. LeBlanc, M. Scott, M. Srinivas, L. Bukys, J. Costanzo, L. Crowl, P. Dibble, N. Gafter, B. Marsh, T. Olson, L. Sanchis; *DARPA Parallel Architecture Benchmark Study*; Prepared for the DARPA Architecture Workshop, November 1986; Butterfly Project Report 13; University of Rochester, Computer Science Department, Rochester, NY, 14627; October 1986

[GJ79] M. R. Garey and D. S. Johnson; *Computers and Intractability: A Guide to the Theory of NP-Completeness*; W. H. Freeman, 1979.

[Ti79] G. Tinhofer; *On the generation of random graphs with given properties and known distributions*; Applied Computer Science (Munich), 13 (1979), pp. 265-297.

[Ul76] J. R. Ullmann; *An Algorithm for Subgraph Isomorphism*; J. ACM, 23 (1976), pp. 31-42.

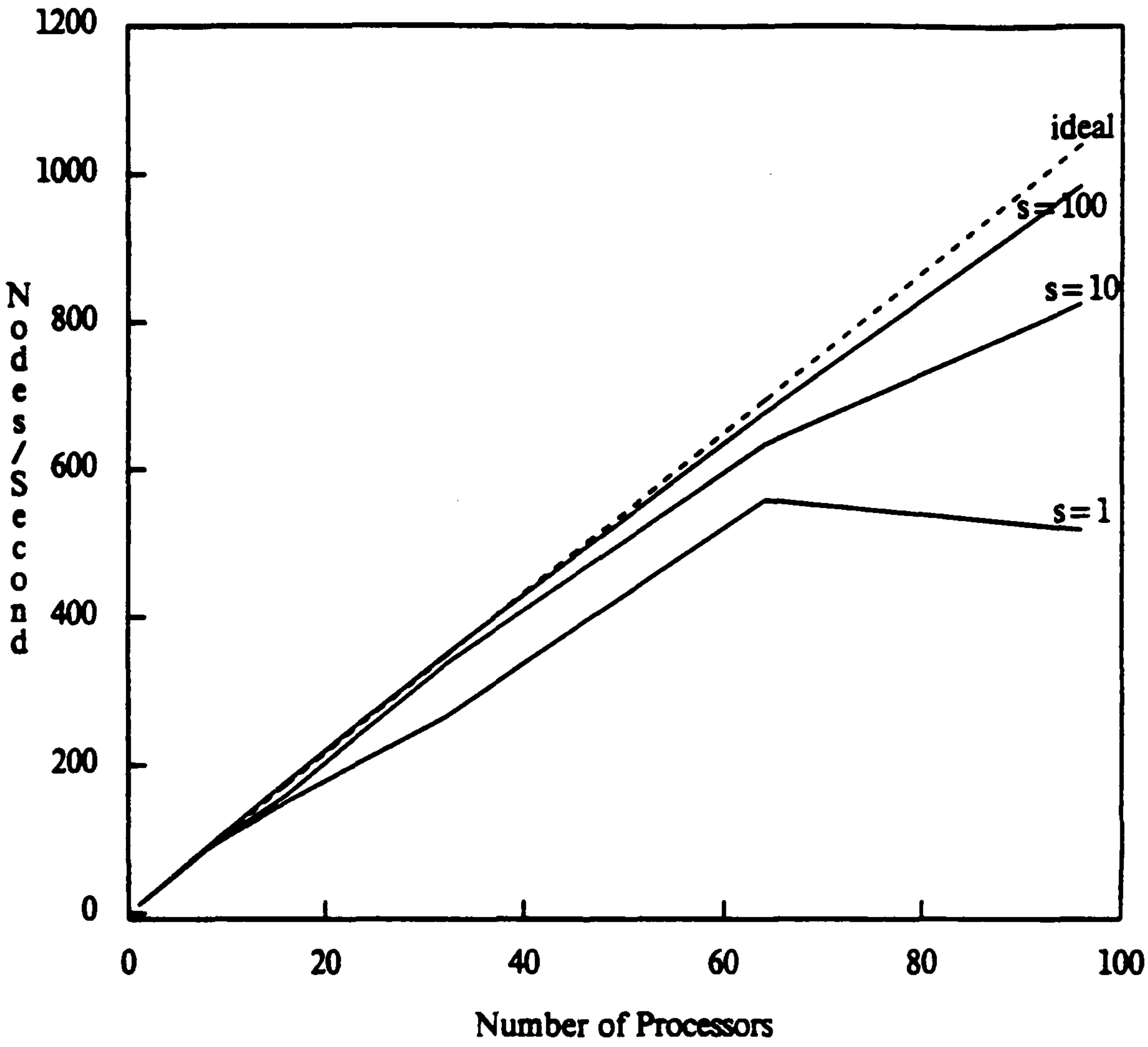


Figure 1:
Node Processing Rate

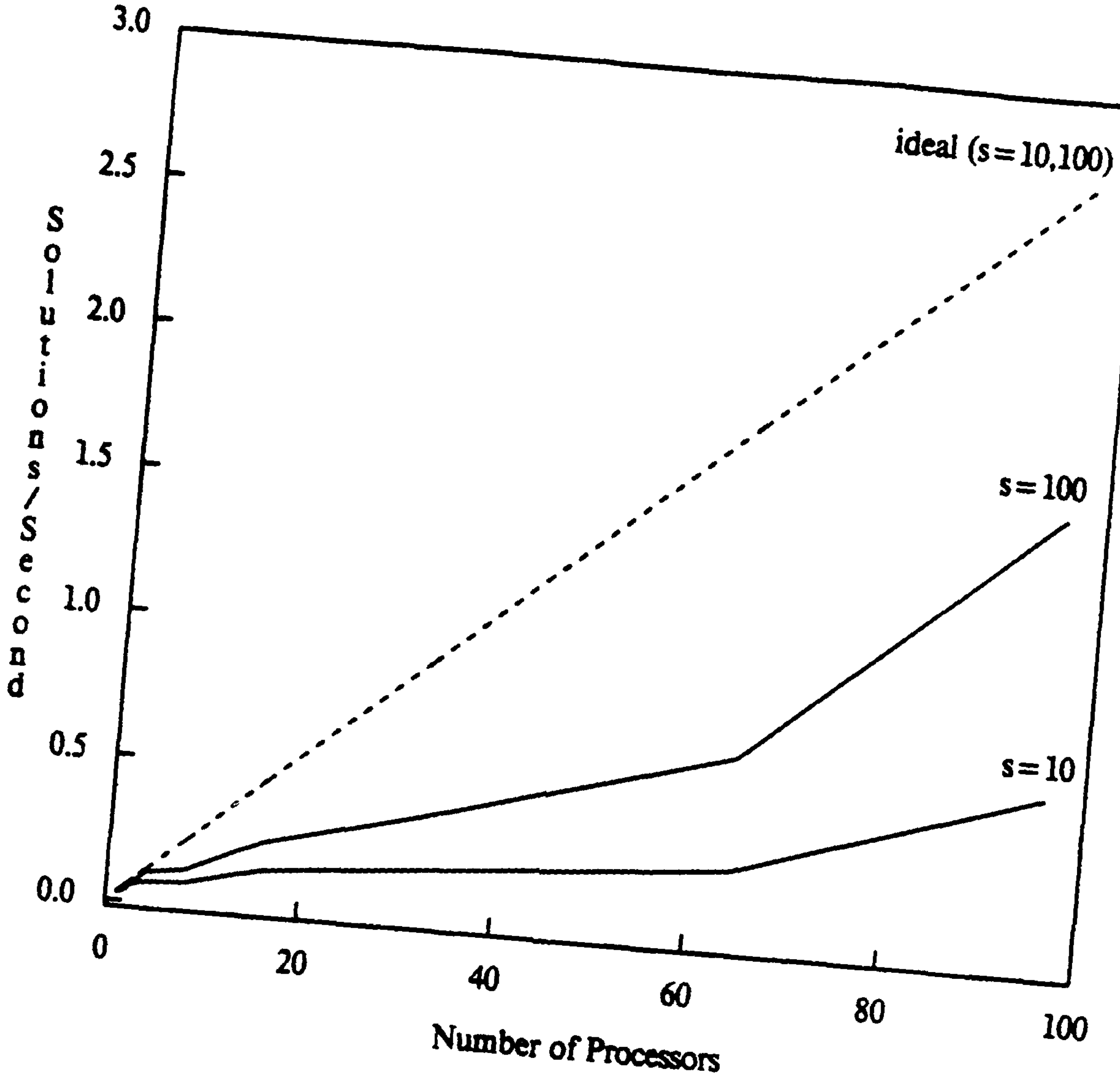


Figure 2:
Solution Rate

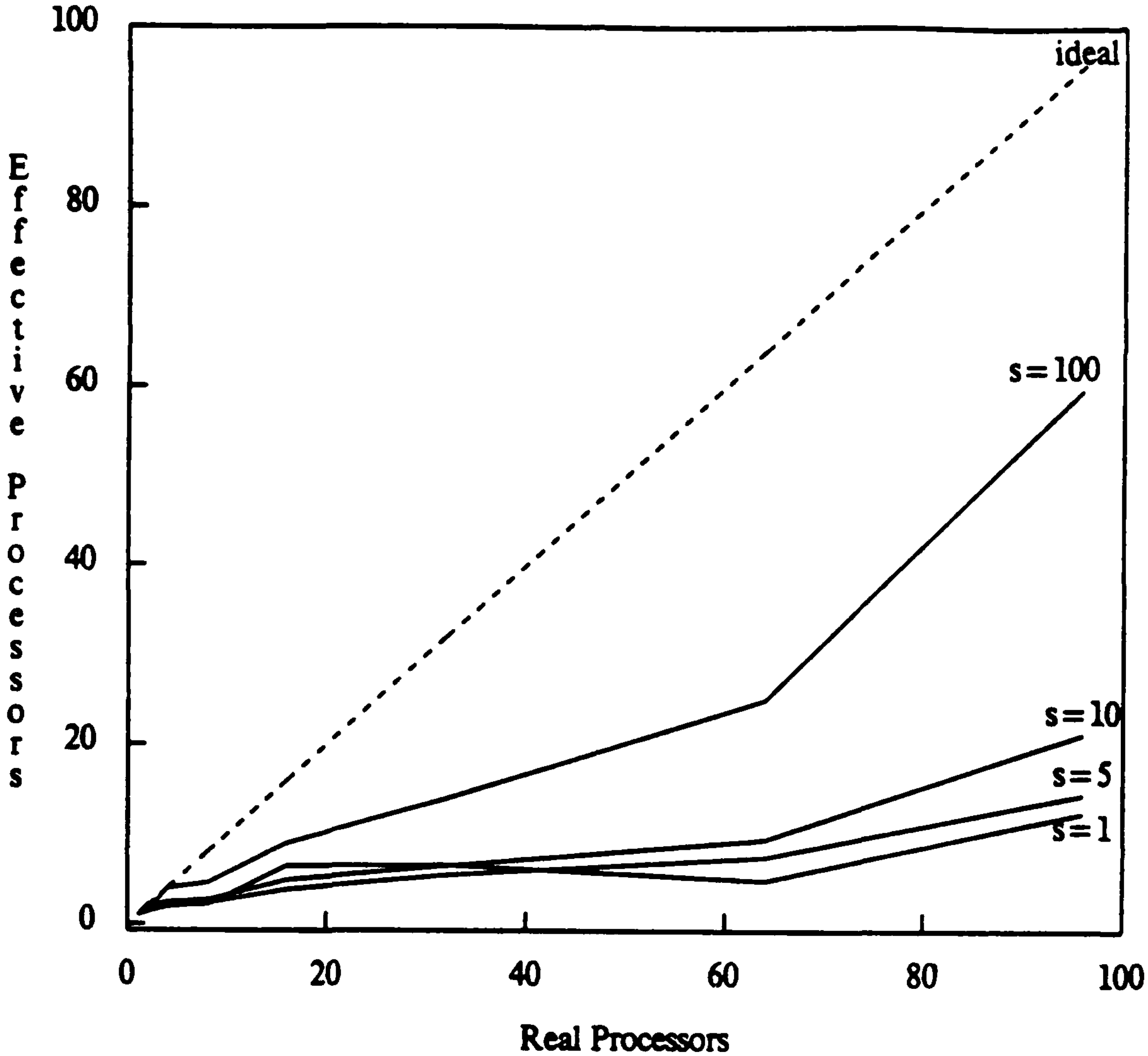


Figure 3:
Processor Efficiency

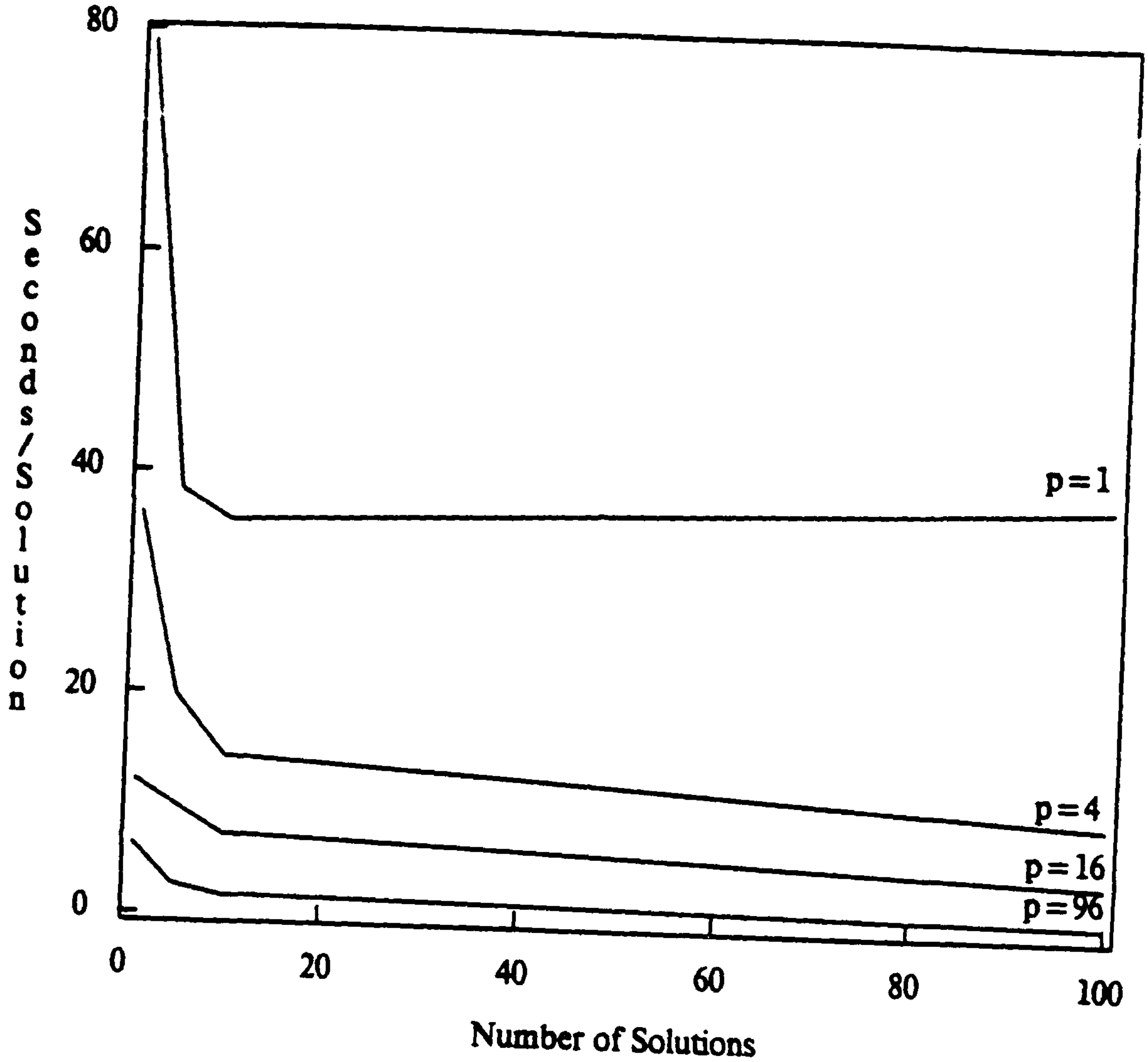


Figure 4:
Cost per Solution versus
Number of Solutions Sought

Appendix B-6

Advanced Likelihood Generators for Boundary Detection

David Sher
Computer Science Department
The University of Rochester
Rochester, New York 14627

January 1987
TR 197

In [Sher85] I discussed the advantages of feature detectors that return likelihoods. In [Sher86] I demonstrated some preliminary results with a boundary point detector that returns likelihoods. Now I have developed boundary point detectors that have these properties:

- Return probabilities
- Can be combined robustly
- Potential sub-pixel precision
- Work with correlated noise.
- Can handle multiple gray-levels (tested with 255)

These detectors were applied both to aerial images and to test images whose boundaries are known. They are compared to established edge detectors.

This work would have been difficult if not impossible without the active support and encouragement of Chris Brown, my advisor, and Jerry Feldman. This work was supported by the Defense Advanced Research Projects Agency U. S. Army Engineering Topographic Labs under grant number DACA76-85-C-0001. Also this work was supported by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB, DC 20332, under Contract No. F30602-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC).

1. Introduction

Currently a great variety of tools are available for low-level vision tasks such as image reconstruction and edge detection. It is time to devote attention to managing tools rather than creating new ones.

Most of the tools for low level vision are algorithms for intermediate steps towards achieving a goal. Here, we consider boundary point detection algorithms in these terms. These are algorithms that try to determine if a boundary passes through a pixel (usually given a window on the image). This task is similar to edge detection. Boundary point detection algorithms do not exist to display outlines pleasing to the human eyes. Their output is meant to be input to a higher level routine such as a shape recognition program or a surface reconstruction program.

Some desiderata for boundary point detection tools are:

(1) The output of a boundary point detector should be useful to as many higher level routines as possible. If every higher level routine required a different boundary point detector then our technology has not lived up to this desiderata.

(2) Boundary point detectors should accept as input the full range of data available. If a boundary point detector only worked for binary data when gray scale data was available, it has not lived up to this desiderata.

(3) Boundary point detectors should do work proportional to the size of the image.

(4) Boundary point detectors should do work proportional to the precision of the output. Thus if subpixel precision is required, it should be made available with work proportional to the required accuracy of boundaries reports.

(5) Boundary point detectors should be parameterized to features of the data. For example, if the distribution of reflectances in the scene is known (the expected image histogram) then a detector should be constructed that uses this information. Another example is if structure in the noise is known (such as correlation) we should be able to take this structure into account.

In this paper I describe an algorithm that fits these desiderata. It is a more advanced version of the algorithm described in [Sher86]. Also the results of tests run on this algorithm are reported here and comparisons with established algorithms such as the Sobel, Kirsch and variants thereupon. Tests will be done soon on more sophisticated operators.

2. Definition of Boundary

Before talking about boundary point detector it is a good idea to define exactly what a boundary is. Vision problems involve imaging a scene. This scene could be an aerial view or a picture of machine parts or an outdoors scene. This scene is filled with *objects*. In an aerial photograph some of these objects are buildings, trees, roads and cars. Each of these objects is projected into the observed image. Thus each object has an *image* that is a subset of the observed image. Where the observed image of one object meets an observed image of another object there is a *boundary*. such boundaries are sometimes referred to as occlusion boundaries.

3. Returning Probabilities

The algorithm I describe fulfills the first desideratum by returning probabilities that a boundary is near a point. Here I justify returning probabilities and show how I can fulfill the first

desideratum using them.

No tool for boundary point detection or any other low-level *vision task ever* does *its task* without a significant probability of being wrong. Thus the error characteristics of a low-level vision algorithm need to be considered. Intermediate-level vision algorithms differ in sensitivities to different kinds of errors.

Consider boundary point detection. Regularization algorithms (interpolation algorithms) [Boult6?] suffer more from missing a boundary point than from having extra ones. When a boundary point is missed regularization algorithms try to smooth over the boundary with disastrous results. Hough transform techniques often work effectively when the set of boundary points detected is sparse because the work a hough transform technique does is proportional to the size of that set.

Another reason that Hough transform techniques do well with sparse data is that they are mode based. Thus Hough transform techniques are robust when feature points are left out and when there are outlying data points. The robustness of the Hough transform is described in detail in [Brown82].

It is good software management to use the same boundary point detector to generate input for all high level vision tasks that require boundary point detection rather than building a special detector for each high level routine. If a boundary point detector returns a true/false decision for each point then its output does not suit both regularization techniques and hough transform techniques. Take for example the one dimensional intensity slice shown in Figure 1.

Figure 1: Slice through an image with an ambiguous boundary

For use as a first stage before regularization it is preferable that such ambiguous slices be considered boundaries because the cost of missing a boundary is high (compared to the cost of missing an edge). For use with hough transform line detection figure 1 is not a good boundary because the cost of an extra edge is high.

The traditional solution to the dilemma of satisfying differing requirements among intermediate level routines has been to supply numbers such as *edge strengths* rather than true/false decisions. These strengths describe how likely the low-level vision algorithm considers the event such as the existence of a boundary. The example in figure 1 has the detector return a low edge strength. Figure 2 shows a 0 edge strength.

Figure 2: Slice through an image with an edge strength of 0

Figure 3 shows a high edge strength.

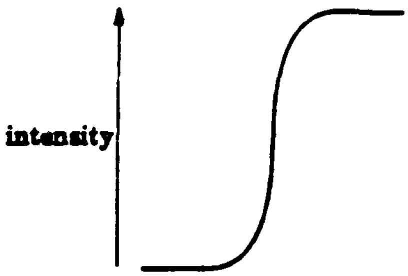


Figure 3: Slice through an image with a high edge strength

If an edge detector that returns strengths (acting as a boundary point detector) is used as input for various intermediate level applications, then each application usually uses a threshold to determine what is and isn't a boundary. If an edge strength is higher than the threshold a boundary is reported. The threshold is determined by running the boundary point detector and the intermediate level algorithm with different thresholds and finding the threshold that results in the best results according to some error norm or human observation[1]. If the boundary point detector is changed new thresholds need to be found.

If boundary point detectors were standardized so that the correspondence between strength and the probability of the boundary were consistent between all boundary point detectors then the threshold need only be calculated once for each intermediate level application. Then the thresholds for intermediate level applications could be calculated from theoretical principles. When the strengths have clear semantics the entire process of threshold determination would be fully understood.

A consistent and well defined output for boundary point detectors is the probability of a boundary. If all boundary point detectors output the probability of a boundary then the boundary point detector could be improved without changing the rest of the system. If the error sensitivities of the intermediate level routines are known the thresholds can be determined by a simple application of decision theory.

4. Likelihoods

Most models for boundary point detection describe how configurations of objects in the real world generate observed data. Such models do not explicitly state how to derive the configuration of the real world from the sensor data. This behavior of models results in graphics problems being considerably easier than vision problems. Thus we have programs that can generate realistic images that no program can analyze.

Given the assumption "There is a boundary between pixels p_1 and p_2" we can determine a probability distribution over possible observed images. Let $b(1,2)$ be the statement that there is a boundary between pixels p_1 and p_2. Let $S(b(1,2))$ be the set of region maps such that $b(1,2)$ is true (and generally I use the notation that $S(\textit{statement})$ is the set of region maps where *statement* is true). Let M be the model for the boundary detection task. Then the probability of O (the observed image) given $b(1,2)$ under M is calculated by equation 1.

[1] Edge relaxation algorithms often adjust the edge strengths. For the purposes of this paper consider such an algorithm as a part of the detector, the output of this detector is the strengths output by the relaxation algorithm.

$$P(O|b(1,2)\&M)=\frac{\sum_{i\in S(b(1,2))} P(O|i\&M)P(i|M)}{P(b(1,2)|M)} \quad (1)$$

If O is the observed data then the probability calculated in equation 1, $P(O|b(1,2)\&M)$, is called here (inspired by the statistical literature) the *likelihood* of $b(1,2)$ given observed data O under M. Generally the models we use in computer vision make the calculation of likelihoods simple for the features we want to extract. However the desired output of a feature detector is the conditional probability of the feature. Thus in boundary point detection I can derive $P(O|b(1,2)\&M)$ and I can derive $P(O|\sim b(1,2)\&M)$ ($\sim x$ means "not x" in this paper) but I want to derive $P(b(1,2)|O\&M)$.

A theorem of probability theory, *Bayes' law*, shows how to derive conditional probabilities for features from likelihoods and prior probabilities. Bayes' law is shown in equation 2.

$$P(f|O\&M)=\frac{P(O|f\&M)P(f|M)}{P(O|f\&M)P(f|M)+P(O|\sim f\&M)P(\sim f|M)} \quad (2)$$

In equation 2 f is the feature for which we have likelihoods. M is the model we are using. $P(O|f\&M)$ is the likelihood of f under M and $P(f|M)$ is the probability under M of f (the prior probability).

For features that can take on several mutually exclusive labels such as surface orientation a more complex form of Bayes' law shown in equation 3 yields conditional probabilities from likelihoods and priors.

$$P(l|O\&M)=\frac{P(O|l\&M)P(l|M)}{\sum_{l'\in L(f)} P(O|l'\&M)P(l'|M)} \quad (3)$$

l is a label for feature f and $L(f)$ is the set of all possible labels for feature f.

Likelihoods are important because they are a useful intermediate term on the way to deriving a posterior probability. An upcoming technical report describes formulas for evidence combination based on likelihoods [Sher87]. To apply that theory of evidence combination one needs to compute the likelihoods explicitly. Thus in the succeeding sections I calculate the likelihoods even up to factors that are equal in all the likelihoods (hence are divided out by equation 3).

Another important use for explicit likelihoods is for use in Markov random fields. Markov random fields describe complex priors that can capture important information. Markov random fields were applied to vision problems in [Geman84]. Likelihoods can be used with a Markov random field algorithm to derive estimates of boundary positions [Marroquin85] [Chou87].

5. Simplifications

To make the problem of computing the likelihoods for events computationally tractable, I simplify my problem in certain ways.

The first simplification is calculating the probability of a boundary at a point rather than trying to compute a probability distribution over boundary maps for the entire scene. Even though a distribution over complete maps would be more general there are too many possible boundary maps to manage realistically. Under certain circumstances all that is needed is to compute the probability of a boundary near each pixel [Marroquin85].

5.1. Window Based Boundary Detectors

Equation 1 implies an algorithm for determining likelihoods of boundaries. It shows that iterating through all the configurations that cause a boundary at a point is a way to find the likelihood of a boundary. A *region map* is a description of where the images of the objects are. Each configuration of objects in a scene implies a region map. However, the set of region maps that contain a boundary between two pixels is too large to iterate through (for a 512 by 512 observed image it is $>>10^{25,000}$). The cardinality of the set of region maps is in general exponential in the size of the image.

An obvious solution to the problem is to reduce the number of pixels. Generally, the further one gets from a boundary the less relevant the data in the image is to that boundary. Thus it is common to use a relatively small window about the proposed boundary for finding the boundary (see figure 4). There are many fewer region maps over a 4 by 4 window than over a 512 by 512 image.

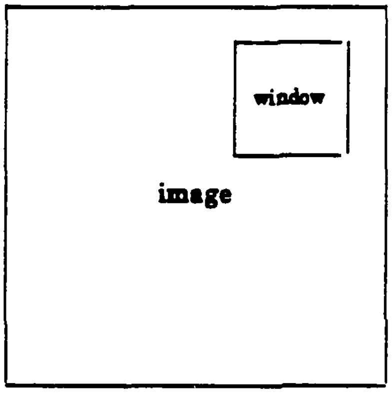


Figure 4: Small Window on Large Image

Thus by looking only at a window I simplify the problem of boundary point detection considerably. Inevitably, I lose some accuracy in the computation of probabilities from limiting the data to a window. However the simplification of the algorithm compensates for this loss. Every attempt at edge detection has used this principle [Hueckel71] [Canny83] with possibly a further stage of linking or relaxation.

5.2. Constraining Region Maps

Applying a boundary point detector to an H (height) by D (depth) observed image involves computing the probability of HD boundaries. If $H=D=512$, then $HD=262144$. A boundary point detection algorithm computes probabilities for all these potential boundaries. The cost of using algorithm that computes the probability of a boundary at a point is multiplied by HD. The cost may be reduced by sharing some of the work between iterations. Still much of the work can not be shared. Saving time by sharing work between iterations is discussed in later sections.

Ignoring saving by sharing between iterations, any saving in the algorithm that computes the probability of a boundary at one point is multiplied manyfold (for $H=D=512$ 262144fold). Algorithms for computing the probability of a boundary at a point require work proportional to the number of region maps. Reducing the number of region maps that need to be considered proportionately reduces the work required by the algorithms for boundary point detection.

The maximal amount a region map can change the likelihood of a feature label is the probability of that region map divided by the prior probability of a feature label corresponding to that region map (shown by a cursory examination of equation 1, repeated below, with i being a region map).

$$P(O|b(1,2)\&M)=\frac{\sum_{i\in S(b(1,2))} P(O|i\&M)P(i|M)}{P(b(1,2)|M)} \tag{1}$$

Bayes' law (equation 3) can be broken into two steps. First the likelihoods are multiplied by the priors. Consider the likelihood of a feature label (say *boundary*) multiplied by the prior probability. Call such a term the *conjunctive probability* of l since it is $P(O\&f=l\&M)$. The conjunctive probability of l can be changed by deleting a region map with $f=l$ at most the probability of that region map. The probability $P(f=l|O\&M)$ is derived from the conjunctive probabilities by dividing $P(f=l\&O\&M)$ by the sum of the conjunctive probabilities $\sum_r P(f=r\&O\&M)$. Thus if the probability of a region map is small compared to $P(O\&M)=\sum_r P(f=r\&O\&M)$ deleting the likelihood corresponding to it has a small effect on the resulting distribution. Thus one can with some safety ignore region maps whose prior probability is small enough.

For standard edge detection algorithms a common restriction on region maps is to assume that there are at most two object images participating in the window thus there can only be two regions [Hueckel71]. Step edge based models implicitly make this assumption [Canny83] [Nalwa84]. Windows with more than two object images in them are assumed to occur infrequently. I call the assumption that there are at most two object images participating in a window the *two object assumption*.

Another simplification places a limitation on the curvature of the boundaries observed in the image. Limiting this parameter limits the set of region maps in a mathematically convenient way. If the curvature is limited enough the boundaries can be considered to be straight lines within windows. Thus the windows on the observed image can be modeled assuming there is no boundary or a single linear boundary across them. A similar model (it allowed two linear boundaries in a window) was used in [Hueckel71]. I call the assumption that there are no high curvature boundaries the *low curvature assumption*.

5.3. Numerical Approximations

Another effect that makes the probabilities calculated by my algorithms inaccurate is that a real number can only be specified to a limited accuracy on the computer. Thus there is a limit to the accuracy that calculations can be performed to in the computer. In section 6 I ignore the error introduced from inexact floating point computations. In my implementation (section 8) I used double precision arithmetic throughout in an attempt to reduce this error.

Another source of errors is my simplifying the mathematics to make the algorithm simpler. One such approximation I make is to use the density of a normal distribution as a probability. In the equations derived in section 6 I use this approximation in the probability derived from a multinormal Gaussian. This approximation simplified the mathematics for deriving the detector. Such an approximation is standard.

6. Building a Boundary Detector

In section 4 I discuss why determining likelihoods is a useful first step in feature detection. In section 5 I describe the approximations and simplifications necessary to make the problem of boundary point detection computationally tractable. Now all that is left is to develop the algorithm. The first step in deriving a boundary point detection algorithm is to derive the likelihood that a window is filled with a single object given the observed data (section 6.1). Then likelihoods for windows with multiple objects are derived (section 6.3). In this section I assume that the model has Gaussian mean 0 noise with a known standard deviation added to an ideal image.

6.1. Likelihoods for a Single Object

The problem is to find the likelihood of a single object filling a window in the image. The expected intensities of the pixels in the window are proportional to the reflectance of O. Let T_0 be the expected value of the pixels in the window when O has reflectance 1, $r(O)=1$. T_0 is often referred to as a *template* in the image understanding literature.

Template matching can be best shown on a two dimensional medium (with weak graphical capacities) when the window is 1 dimensional. Thus I use a 1 by 8 window for examples. A typical template for a single object in a 1 by 8 window is shown in figure 5.

100	100	100	100	100	100	100	100

Figure 5: Template for a single object

This template is boring because I assume that there is little variance in intensity in the image of the interior of an object. The observed image of the object has a normal iid added to each pixel. Thus the observed image (when the standard deviation is 8) can look like figure 6.

94	104	100	94	92	105	101	103

Figure 6: Noised Template for a single object

The probability that the noised window results from the template is a function of the vector difference between the window and the template. For the example in figure 6 given the template for a single object in figure 5, the probability is calculated by summing the squared differences between the template and the observed data (187) and then applying the normal distribution function (for 8 independent normally distributed samples mean 0 standard deviation 8 rounded to the nearest integer) to get 8.873e-12.

In later sections when I use windows or templates in equations the windows or templates are flattened into vectors. Thus a two dimensional window is transformed into a vector (figure 7).

1	2	3
4	5	6
7	8	9

Window 1

1	2	3	4	5	6	7	8	9

The Vector from Window 1.

Figure 7: Flattening a window into a vector

All the windows and templates are assumed to be flattened thus. When a template is subtracted

from a window, vector subtraction is happening. When a template or window is being multiplied by a matrix, vector matrix multiplication is happening. In particular, WW^T is the sum of the squares of the elements of W while WT^T is the sum of the products of the corresponding elements of W and T.

Let aT_θ be a times the intensities in the template T_θ. The template for an object with reflectance $r(O)$ is $r(O)T_\theta$. The probability of observing a window W when the scene is of O is determined by the Gaussian additive factor. Equation 4 is the formula for the probability. (Let V be the variance of the noise, σ^2 in the rest of this paper) Let n be the size of the window and κ be the constant $1/(2\pi V)^{n/2}$.

$$P(W|r(O)=r\&M)=\kappa\exp(-(W-rT_\theta)(W-rT_\theta)^T/2V) \tag{4}$$

Since the reflectance of the object in the scene is not known but the distribution of reflectances is known one must integrate this formula over possible reflectances. Thus the probability of a window given it is of a single object of known position and shape is in equation 5.

$$P(W|O\&M) = \int\kappa\exp(-(W-rT_\theta)(W-rT_\theta)^T/2V)dD_r(r) \tag{5}$$

The term $(W-rT_\theta)(W-rT_\theta)^T$ can be rearranged to $WW^T-2rT_\theta W^T+r^2T_\theta T_\theta^T$. WW^T is the sum of the squares of the pixels in the window. Let us refer to it as W^2 for shorthand. $T_\theta W^T$ is the correlation between T_θ and W. Let us refer to it as C for shorthand. $T_\theta T_\theta^T$ is the sum of the squares of the elements of the template. Let us refer to it as T^2. Using a rearrangement and these shorthands equation 5 can be restated as equation 6.

$$P(W|O\&M) = \exp(-W^2/2V)\kappa\int\exp((2rC-r^2T^2)/2V)dD_r(r) \tag{6}$$

Equation 6 is the product of two parts, equation 7 :

$$F_1(W^2) = \exp(-W^2/2V)\kappa \tag{7}$$

and equation 8 :

$$F_2(C) = \int\exp((2rC-r^2T^2)/2V)dD_r(r) \tag{8}$$

F_1 and F_2 have one parameter that depends on the window (T^2 is unchanged over all windows). They can be implemented by table lookup without excessive use of memory or much computation.

Hence the algorithm to calculate the likelihood of a window given it is of a single object of known position and shape (but unknown reflectance) is described by figure 8 below.

Figure 8: Algorithm to Calculate Likelihood of Known Object for a Single Window

			Multiplies	Adds
W^2	:=	WW^T	w	$w-1$
C	:=	WT_θ^T	w	$w-1$
Output		$F_1(W^2)*F_2(C)$	1	0

w is the number of pixels in W and also the number of pixels T_θ. Figure 8 is a $4w-1$ operations algorithm.

6.2. Reducing the Configurations Considered

In section 4 I describe how to derive the likelihood of a boundary at a point from the likelihoods of configurations of a the scene given a boundary at a point (equation 1). Here I justify

using a small number of configurations of the scene. Having reduced the set of configurations I can derive an efficient algorithm for generating likelihoods given multiple objects. In section 5.1 I made the assumption that only a small window on the image need be looked at. In section 5.2 I justified ignoring region maps with low probability.

There is some number of objects, N_O, such that the probability N_O or more objects having an image in a single window has a probability much smaller than the probability of all but a small probability subset of the region maps. Hence I need only consider configurations of $N_O - 1$ or less objects in a window. The two object assumption of section 5.2 is equivalent to saying $N_O = 3$.

There are still a large set of configurations of less than N_O objects. Consider the ideal image given such a configuration and some coloring of objects. Consider a window on this ideal image, W_I. There is a set of configurations with the same coloring such that the resulting window on the ideal window from such a configuration W_J such that:

$$(W_I - W_J)(W_I - W_J)^T < \varepsilon \tag{9}$$

The likelihood that the observed image results from any of the configurations that fit the criterion of equation 9 and coloring is close to the likelihood computed from W_I because if W is the observed window the likelihood is a function of the norm of $W - W_J$. Thus for efficiency sake we can consider the likelihood of each configuration and coloring that fit equation 9 the same as that of the configuration and coloring of the template that generates W_I. Hence we can only consider a small set of configurations of objects. With a similar argument one can prove that only a subset of the possible colorings need be considered. Hence I can justify using an argument of this sort using a small set of templates and objects. How much inaccuracy results from these simplifications can be analyzed mathematically.

A step edge model can be derived by assuming that objects' images have a uniform intensity and the two object assumption. Thus such simplifications underlies work like that of [Hueckel73] [Canny83]. More sophisticated assumptions about the intensities of objects images result in more complex models [Binford81] that still can be reduced to a reasonable number of configurations reflectances with a corresponding loss of accuracy in the resulting probabilities.

6.3. Algorithm for Likelihoods of Multiple Objects

Here I derive an algorithm for finding the likelihood that a scene that contains several objects given a window.

The statement that the configuration of objects is near the specified configuration is called C. C has N_O objects O_n with reflectances r_n. Associated with C are N_O templates T_n. Each T_n is the light that hits the image given that O_n has reflectance 1 and unit lighting.

The template that represents the expected window when the O_n have reflectances r_n is $\sum_n r_n T_n$. Hence the likelihood of the window when the objects have reflectances r_n is shown in equation 10.

$$P(W \mid C \,\&\, r(O_1) = r_1 \cdots r(O_n) = r_n \,\&\, M) = \kappa \exp(-(W - \sum_n r_n T_n)(W - \sum_n r_n T_n)^T / 2V) \tag{10}$$

To derive the likelihood of the window when the objects are of unknown reflectance it is necessary to integrate over all possible sets of r_n. Equation 11 shows the integrated equation.

$$P(W|\mathbf{C}\&M) = \kappa \int \exp(-(W - \sum_n r_n T_n)(W - \sum_n r_n T_n)^T / 2V) dD_e r_n \tag{11}$$

I use the simplifying notation from section 6.1. Also let C_n be WT_n^T. Then equation 11 can be transformed into equation 12.

$$P(W|\mathbf{C}\&M) = \kappa \exp(-W^2 + w\alpha^2/2V) \int \exp(\sum_n r_n C_n / V) \exp(-(\sum_n r_n T_n)(\sum_n r_n T_n)^T / 2V) dD_e r_n \tag{12}$$

As in section 6.1 I can break up equation 12 into a pair of functions F_1 and F_4. F_1 is as before F_4 is described by equation 13.

$$F_4(C_1, C_2, \cdots C_{N_O}) = \int \exp(\sum_n r_n C_n / V) \exp(-(\sum_n r_n T_n)(\sum_n r_n T_n)^T / 2V) dD_e r_n \tag{13}$$

Unlike F_3, F_4 is a function of several variables. Thus using a table for table lookup on F_4 takes a large amount of memory since an entry must be made available for each possible value of the elements. Experiments with constructing F_4 tables for 2 object configurations show that F_4 is smooth and near quadratic. As an example F_4 for standard deviation 12 noise and 5 by 5 templates is plotted in figure 9.

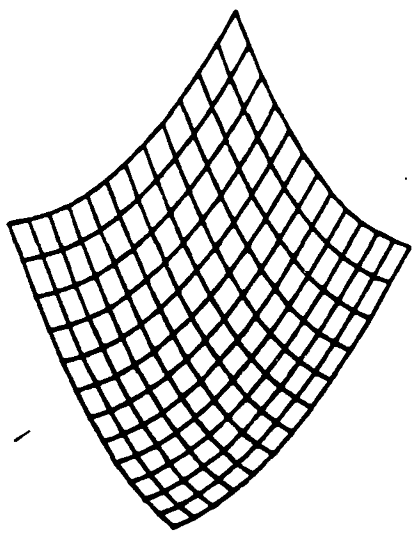


Figure 9: log(F_4) with stdev 12 noise and 5x5 template

Hence the tables can be stored sparsely and splines or even linear interpolation used to get at values that were not given without introducing serious inaccuracy. If the table is close enough to a quadratic function then perhaps only a polynomial need be stored.

If all the T_n are orthogonal (their pairwise vector products are 0) and D_c is a probability distribution where colors of objects are uncorrelated then F_4 can be partitioned into a product of N_O functions. Each of these functions computes the probability that the relevant part of the window observed data is the image of O_n for some n. Here N_O large precise tables can be stored to compute each of the functions and their product used for F_4.

Since F_4 can be calculated there is a feasible algorithm for determining the likelihood of an observed image given a particular pair of objects. A description of that algorithm and the times spent in each step is shown in figure 10.

Figure 10: Algorithm to Calculate Likelihood of Multiple Objects for a Single Window

			Multiplies	Adds
W^2	:=	WW^T	w	$w-1$
N_O times C_i	:=	WT_i^T	$N_O w$	$N_O w - N_O$
Output		$F_1(W^2)^* F_4(C_1,C_2)$	1	0

I did not consider the cost of the interpolation or splining for F_4 in the operation counts in figure 10. The cost of this algorithm is $(N_O+1)w+1$ multiplies and $(N_O+1)(w-1)$ adds plus the cost incurred by the interpolation. This cost occurs for each window.

6.4. Blurred Images

The previous sections assume that the only degradation of the image data is a result of adding a normal random variable to each element of the image (noise axiom). However lenses and many other sensors degrade the image through blur. Motion also causes blur on film and other sensors. Blur is often modeled as a linear transformation of the image data that is applied before the normal variable is added to the pixels [Andrews77]. When blur is shift independent (same blur everywhere in the image) then the linear operator must be a convolution operator.

The algorithms specified in sections 6.1 and 6.3 require changes to work under this new assumption. If the blur is linear the change required to these algorithms is to use blurred templates. The likelihood of the observed data given a template is a function of the vector difference between the expected observation without the normal additive iid and the observed data. If it is expected that a blurring has happened then one need to use a blurred template instead of the unblurred template used in the previous algorithms.

The algorithm of section 6.3 uses two templates (one for each object). If the blur is linear then the linear function aT_1+bT_2 and the blur function commute. Thus the two templates can be blurred and the result of correlating the window with the two blurred templates can be used with this algorithm.

The problem I address in the rest of this section is how to compute a blurred template from an unblurred template. A shift independent blurring operation is applying a convolution operator to the image. A convolution operator has limited extent if the illuminance of at a pixel in the blurred image depends on a window in the unblurred image. Such a convolution operator can be described by a matrix the size of the window that describes the effect each point in the window has on that point of the blurred image. So a blurring function that causes each point of the blurred image to be the result of .5 from the corresponding point of the unblurred image and .25 from the points immediately to the right and the left has a matrix described by figure 11.

.25	.5	.25

Figure 11: Simple Blurring Function Matrix

Given a blurring function matrix of size (M_w, M_l) and an unblurred template of size (T_w, T_l), a blurred template of size $(T_w - M_w + 1, T_l - M_l + 1)$ can be calculated (figure 12) ($\otimes$ means convolution).

T:

100	100	100	200	200	200

$\otimes$

M:

.25	.5	.25

=

$T \otimes M$:

100	125	175	200

Figure 12: Effect of Blur Matrix M on Template T

To develop a larger blurred template than $(T_w - M_w + 1, T_l - M_l + 1)$ requires that the blur function be applied to points outside the unblurred template. If the expected values for such points are derived then a larger unblurred template has been constructed. Hence the derivation of a smaller blurred template from an unblurred template suffices for the construction of blurred templates.

6.5. Correlated Noise

The previous sections assume that an uncorrelated normal variable called noise that is added into the illuminance of each pixel in the ideal image to get the observed image. It is possible to relax the assumption that the noise added to each pixel is uncorrelated with the noise added to the other pixels. Instead a matrix C can be supplied that describes the correlation between the noise variables of the window.

One problem is how to handle correlations between points in the window and points outside the window. Since one can only correlate with expected values of points outside the window (since we chose to ignore the data from such points in our calculations) the effect of such points can only introduce a constant factor into the likelihood calculations. When the likelihoods are converted into probabilities this constant factor is divided out. Hence I can safely ignore such a constant factor. For the purposes of evidence theory I may need to derive the constant factor but it need only be derived once and then all the likelihoods be only multiplied by it.

The algorithm in section 6.3 has the algorithm in sections 6.1 as a special case. Thus if I derive the algorithm corresponding to the one in section 6.3 I can derive the other algorithm. If I have window W and I expect (possibly blurred) templates T_n with unknown reflectances then the equation that describes the likelihood of W is equation 14.

$$P(W|O_1 \cdots O_n \& M)$$
$$=$$
$$\kappa \int \exp(-(W - \sum_n r_n T_n)C(W - \sum_n r_n T_n)^T/2) dD_\epsilon r_n \tag{14}$$

I introduce notation to simplify equation 14 to the point where an algorithm naturally derives from it. Let W_C^2 be WCW^T. C is symmetric so let $c_n = WCT_n^T = T_1CW^T$. Then equation 14 can be rearranged and simplified to equation 15.

$$P(W|O_1 \cdots O_n \& M)$$
$$=$$
$$\kappa \exp(-W_C^2/2) \int \exp(\sum_n r_n c_n) \exp(-(\sum_n r_n T_n)C(\sum_n r_n T_n)^T/2) dD_\epsilon r_n \tag{15}$$

I can then describe equation 15 as the product of two functions, F_5 that takes W_C^2 as an argument and F_6 that takes the set of c_n as arguments. Equations 16 describe F_5 and F_6.

$$F_5(X) = \kappa \exp(-X/2)$$
$$F_6(X_1, \cdots X_{N_O}) = \int \exp(\sum_n r_n X_n) \exp(-(\sum_n r_n T_n)C(\sum_n r_n T_n)^T/2) dD_\epsilon r_n \tag{16}$$
$$P(W|O_1, \cdots O_{N_O} \& M) = F_5(W_C^2) F_6(c_1, \cdots c_{N_O})$$

Equation 16 is simple enough to derive an algorithm that calculates the likelihood of a window given a template and correlated noise with standard deviation σ and correlation matrix C. Figure 13 shows this algorithm.

Figure 13: Algorithm to Calculate Likelihood of Multiple Objects with Correlated Noise

			Multiplies	Adds
W_C^2	:=	WCW^T	$w(w+1)$	$(w+1)(w-1)$
N_O times c_2	:=	WCT_n^T	$N_O w$	$N_O(w-1)$
Output		$F_5(W_C^2)^* F_6(c_1, c_2)$	1	0

Like F_4, F_6 may require interpolation. The cost of the potential interpolation was not figured into these calculations. The algorithm with correlation between the noise variables requires $w(w+1)+N_O w+1$ multiplies and $(w+1)(w-1)+N_O(w-1)$ adds. Substantial savings may be found when C is sparse. Correlation matrices are typically band matrices. If there are b bands in C then the number of multiplies is less than $(b+1)w+N_O w+1$ and the number of adds is less than $(b+1)(w-1)+N_O(w-1)$ adds.

6.6. Sharing the Work

The algorithms in figures 8, 10 and 13 are algorithms for finding the likelihood of a particular template for a single window. Many of an observed image's windows overlap. If likelihoods are being computed for two overlapping windows much of the work in computing the likelihoods can be shared between the computations on the two windows. If the likelihoods are being computed for every window on the image such savings can be substantial.

When taking the sum of the elements of two overlapping windows, as is one in the algorithm of figures 8 it is necessary to only sum the overlap once. Figure 14 gives an example of this

savings.

20	10	15	19
13	18	13	16
9	18	11	4

W_1 W_2

$$\Sigma(W_1 \cap W_2) = 10+15+18+13+18+11 = 85$$

$$\Sigma(W_1) = 20+13+9+\Sigma(W_1 \cap W_2) = 127$$

$$\Sigma(W_2) = \Sigma(W_1 \cap W_2)+19+16+4 = 124$$

Figure 14: Summing the elements of two overlapping windows

The work in summing the squares of the elements in two windows can be shared this way too. If the likelihood generator is being used on every window on an image then the work needed to calculate sums and sum of squares is a fraction of that needed to calculate the same statistics for the same number of non-overlapping windows.

If every window (or a substantial fraction thereof) of an image has the algorithms in figure 8 or 10 run on them then the work involved in convolving the image with templates can be saved too (at least when the templates grow large). Convolution can be performed with the fast Fourier transform at substantial saving in operations for large templates. For algorithm 13 when the correlation matrix has structure (such as being a band matrix) then the fast Fourier transform can be used with substantial savings too.

Thus much of the work can be shared when likelihoods are being determined for every window of an image. Hence the likelihood generators described in figures 8, 10, and 13 are competitive in speed with most standard edge detections schemes.

Another way the work can be shared is that some of the templates used that describe object configurations in a window is by describing the configuration in another template shifted 1 pixel over (see figure 15).

Figure 15: T_1 is T_2 shifted 1 pixel to the right

If every window in the image is being processed then the likelihoods corresponding to the template T_2 are approximated by the likelihoods calculated by the template T_1 for the window 1 pixel to the right. To realize why such an approximation is good, consider that using a window is itself an

approximation. The likelihood of the configuration of objects described by T_1 is approximated by running the algorithm over a window. The likelihood of the configuration described by T_2 can be approximated by running the same algorithm over a window shifted 1 to the right.

Thus the likelihoods for the template corresponding to T_2 need only be calculated for the windows on the far right hand side of the image (the other windows have the T_1 template run on them already). Thus instead of having to take into account templates corresponding to the same configuration of objects shifted several pixels in some direction one need only use a single template and use the output from this template on windows shifted in that direction.

6.7. Getting Probabilities from Likelihoods

Given likelihood generators the remaining task is to calculate probabilities from these likelihoods (using priors). The first task that needs to be done is to group the likelihoods generated into sets that support different labelings for the features. Thus if the configurations C_1, C_2, C_3 correspond to the existence of a boundary at a point and C_4, C_5 and C_6 represent situations that aren't boundaries at that point then I must collect the likelihoods based on C_1, C_2 and C_3 into a single likelihood and similar with the likelihoods collected from C_4, C_5 and C_6. Then I would have a likelihood corresponding to each possible labelings of my feature (for boundary point detection I need to determine the likelihoods corresponding to the existence of a boundary and those that correspond to the nonexistence). Given these likelihoods I can use Bayes' rule to derive probabilities (see equation 3).

The likelihood of a boundary is the probability of the observed scene being generated when an object configuration corresponding to the existence of a boundary exists. I can derive an equation for calculating the probability of a boundary from the outputs of my likelihood generators if I have the prior probability that the configuration is the position of the objects in the scene for each configuration that corresponds to a boundary. Let the set of configurations that correspond to a boundary be represented by $\mathbf{C}_b$. Equation 17 is the first step in the derivation expanding out the likelihood into conditional probabilities.

$$P(W_O|\mathbf{C}_b)=\frac{P(W_O\mathbf{C}_b)}{P(\mathbf{C}_b)} \tag{17}$$

Since the real scene can not correspond to two different configurations I can expand equation 17 into equation 18.

$$P(W_O|\mathbf{C}_b)=\frac{\sum_{c\in\mathbf{C}_b} P(W_O\&c\in\mathbf{C}_b)}{\sum_{c\in\mathbf{C}_b} P(c\in\mathbf{C}_b)} \tag{18}$$

A slight change to equation 18 introduces the likelihoods generated by the algorithms in figures 8 through 13 and the prior probabilities that the scene is in a configuration tested by the algorithms. Equation 19 shows this change.

$$P(W_O|\mathbf{C}_b)=\frac{\sum_{c\in\mathbf{C}_b} P(W_O|c\in\mathbf{C}_b)P(c\in\mathbf{C}_b)}{\sum_{c\in\mathbf{C}_b} P(c\in\mathbf{C}_b)} \tag{19}$$

Equation 19 allows me (given priors on the templates or template sets) to gather many likelihoods into a single one. If I have likelihoods for every feature label and prior probabilities that the feature takes on that label I can use Bayes' law as in equation 3 (reprised here) to derive probabilities for feature labels given the data in the window.

$$P(l_f|O\&M)=\frac{P(O|l_f\&M)P(l_f|M)}{\sum_{l'_f \in L_f} P(O|l'_f\&M)P(l'_f|M)} \tag{3}$$

6.8. Estimating Boundaries

In section 6.7 I show how to derive probabilities given a likelihood generator. Often one must use programs (e. g. programs supplied as part of a package) that take as input estimates of the positions of the boundaries. Such programs can not use probabilities, they just want a boundary map. Here I show how to generate such an input.

To estimate where the boundaries are in an image it is necessary first to develop a cost function that describes what costs errors in estimation have. To use the probabilities of boundaries at points to estimate the configuration of boundaries in an image optimally it is necessary to use a cost function that sums the effects of pointwise errors. Such cost functions are simple to understand and require few parameters to describe (namely only the costs of different mislabelings at a point). I only use this type of cost function in this part of the paper. I also assume that making a correct decision has 0 cost.

For boundary point detection the costs that need to be calculated are:

(1) the cost of labeling a point as a boundary when there is no boundary there.

(2) the cost of labeling a point as not being a boundary when it is.

Call the cost of labeling a point (x,y) as a boundary point when it isn't $c_1(x,y)$ and the cost of labeling a point as not being a boundary when it is $c_2(x,y)$. Let $p_B(x,y)$ be the probability of a boundary at (x,y). Let $e_B(x,y)$ be 1 when the estimation procedure indicates there is a boundary at (x,y) and 0 otherwise. We want a detector that minimizes the expected cost for the estimation. Thus we want to minimize the summation in equation 20.

$$\sum_{(x,y)} c_1(x,y)(1-p_B(x,y))e_B(x,y)+c_2(x,y)p_B(x,y)(1-e_B(x,y)) \tag{20}$$

Let us assume that $c_1(x,y)$ is the same for all (x,y) and the same for c_2. Equation 20 is clearly minimized by minimizing equation 21 for each (x,y) ((x,y) is deleted for clarity).

$$c_1(1-p_B)e_B+c_2p_B(1-e_B) \tag{21}$$

Equation 21 can be rearranged into equation 22.

$$(c_1(1-p_B)-c_2p_B)e_B(x,y)+c_2p_B \tag{22}$$

Clearly you want e_B to be 1 when equation 23 is positive and e_B to be 0 when equation 23 is negative.

$$c_1(1-p_B)-c_2p_B \tag{23}$$

This statement can be algebraically transformed into the statement that e_B should be 1 when the inequality in equation 24 is satisfied and 0 otherwise.

$$p_B < \frac{c_1 - c_2}{c_1} \tag{24}$$

Thus one only needs to threshold the probabilities of boundaries with $\frac{c_1 - c_2}{c_1}$ to estimate the positions of the boundary for the additive cost function with costs c_1 and c_2. This argument is standard in Bayesian decision theory with simple loss functions [Berger80].

7. Implementation Details

Here, I describe my implementation of the algorithms described in section 6. I have code for the algorithms in figures 8, and 10. I also have constructed the code that is implied by equations 19 and 3 of section 6.7.

7.1. Likelihood Generators

These algorithms are based on the assumption that the scene can be modeled by a set of templates. The templates are objects of unit reflectance under unit lighting. I have one template that represents the window being in the interior of the object shown in figure 16.

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Figure 16: Template for the Interior of an Object

For each of 4 directions, 0 degrees, 45 degrees, 90 degrees, and 135 degrees, I have 3 pairs of templates that describe three possible boundaries. For example figure 17 shows the 0 degree templates.

1st pair

1	1	0.5	0	0
1	1	0.5	0	0
1	1	0.5	0	0
1	1	0.5	0	0
1	1	0.5	0	0

0	0	0.5	1	1
0	0	0.5	1	1
0	0	0.5	1	1
0	0	0.5	1	1
0	0	0.5	1	1

2nd pair

1	1	0	0	0
1	1	0	0	0
1	1	0	0	0
1	1	0	0	0
1	1	0	0	0

0	0	1	1	1
0	0	1	1	1
0	0	1	1	1
0	0	1	1	1
0	0	1	1	1

3rd pair

1	1	1	0	0
1	1	1	0	0
1	1	1	0	0
1	1	1	0	0
1	1	1	0	0

0	0	0	1	1
0	0	0	1	1
0	0	0	1	1
0	0	0	1	1
0	0	0	1	1

Figure 17: Template for the 0 degree Boundary

Each pair of templates represent two objects, one occluding the other. I have not generated any templates for windows with 3 objects.

The algorithms in figures 8, and 10 consist of a part that is dependent on a template used and a part that is a function of the observed window. It is the part that depends on the template that presents implementation difficulties. Function F_2 is a function of the sum of squared elements in the template. Function F_4 is a function of the pairwise products of the templates representing the objects in the configuration. Both of these functions were implemented by table lookup with linear

interpolation in my implementation. For every direction the sum of squares of the templates and the pairwise products are described by the same 5 numbers. Only two F_4 tables need to be generated.

F_2 and F_4 also depend on the standard deviation of the noise in the image. I call a likelihood generator that assumes a specified standard deviation of noise a likelihood generator *tuned* to that standard deviation of noise. I have likelihood generators tuned to noise with σ equal to 4, 8, 12, and 16.

7.2. Probabilities from Likelihoods

I use equation 19 to gather together the three likelihoods generated from the 3 pairs of templates in each direction into a single likelihood. Thus I have the likelihoods for the four directions that a boundary passes through or next to the center pixel.

I also need the likelihood that there is no boundary near or through the center pixel. To get this likelihood I take the likelihood that the window is in the interior of the object and combine it with likelihoods for central boundaries calculated for the neighboring windows. Thus from the 0 degree boundary likelihoods I use the likelihood from the window one pixel left and right and combine it with the likelihood of a noncentral edge.

Thus I have 4 likelihoods for 4 directed boundary points and one likelihood that represents the likelihood that there is no central edge in the image. I then use Bayes' law from equation 3 to compute the probabilities of these 4 states. I then threshold the probabilities at 0.5 to present the results shown in section 8. Throughout I assumed that the prior probability of a central edge is 0.1 and that this probability is equally distributed in all 4 directions. This prior information is sufficient to apply Bayes' law.

8. Results from Implemented Boundary Detectors

I have implemented the algorithms in sections 6.1 and 6.3 figures 8 and 10. Here I describe the results from testing this detector.

The software I have written is flexible. However I have only constructed templates for a restricted set of configurations. I have templates for a step edge model with the low curvature assumption, 4 possible orientations for boundaries, and 255 gray levels in the image. My templates handle boundaries that occur in the center of pixels or between pixels. I have built the templates for a 5x5, 7x7 and 9x9 windows. I also have constructed tables to compute F_1 and F_4 for each of these windows that assume noise of standard deviations 4,8,12, and 16.

8.1. Results with Artificial Images

I have applied these operators to test images constructed by a package of graphics routines. This package was written by Myra Van Inwegen and is described in an upcoming technical report.

I describe my operators applied to two test images generated by this package. One is an image of two circles shown on the left in figure 18a.

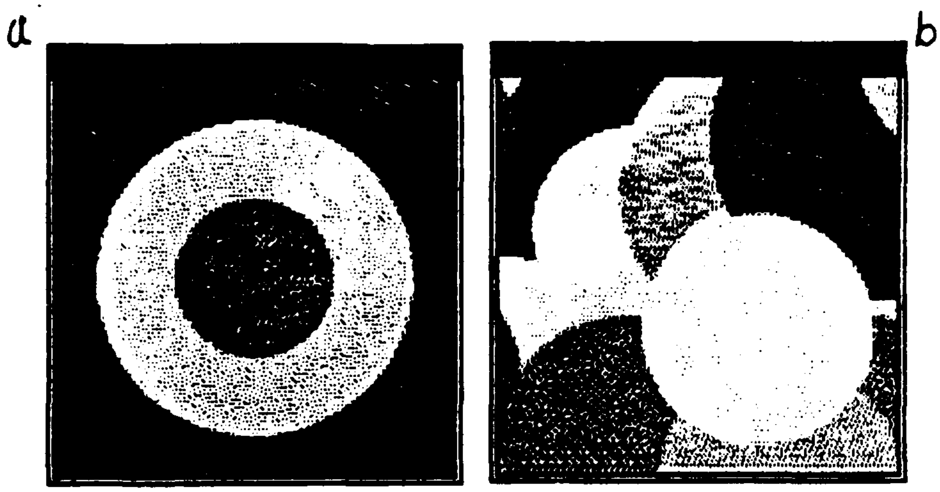
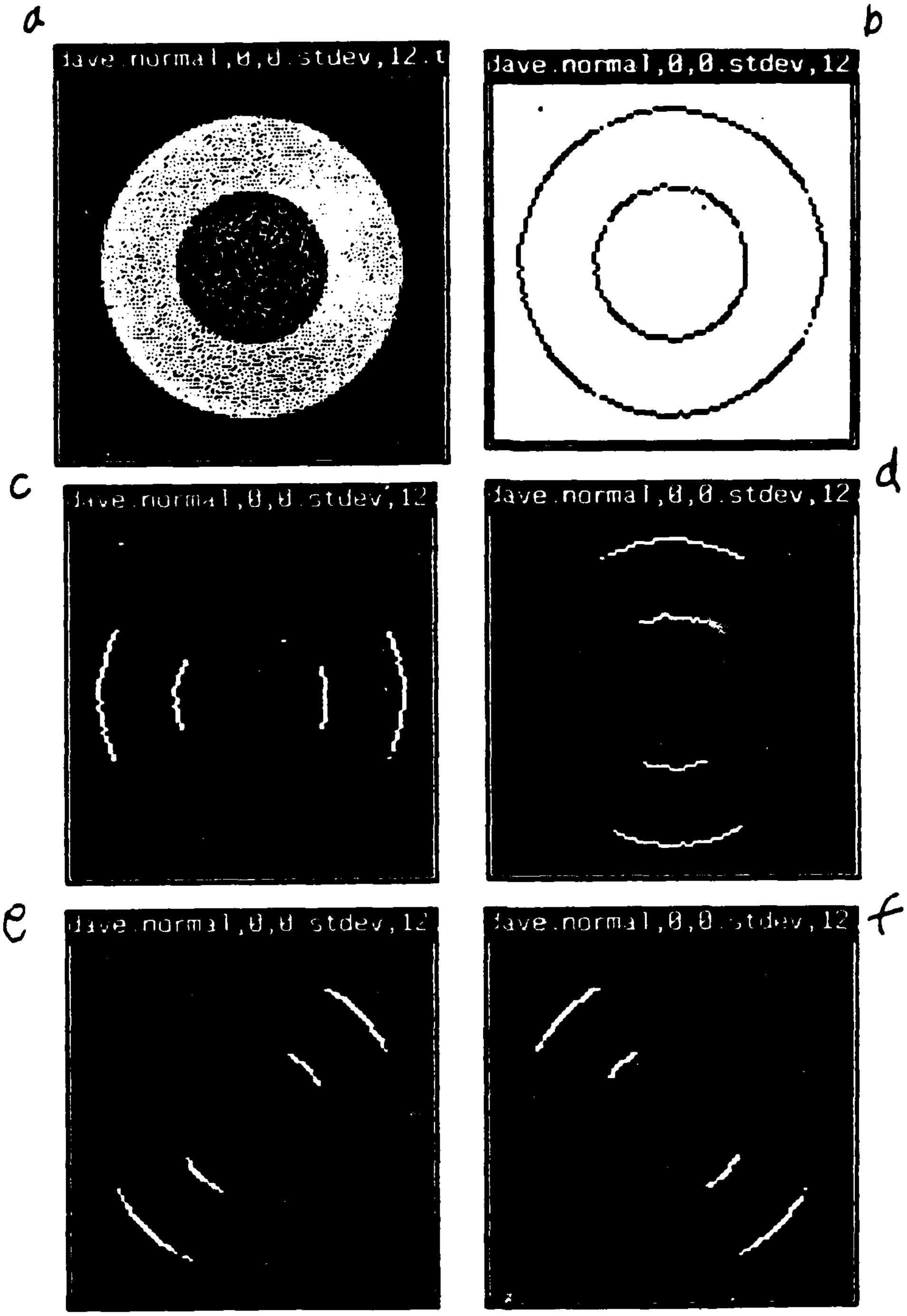


Figure 18: Artificial Images

A more challenging and complex image has also been tested shown in figure 18b.

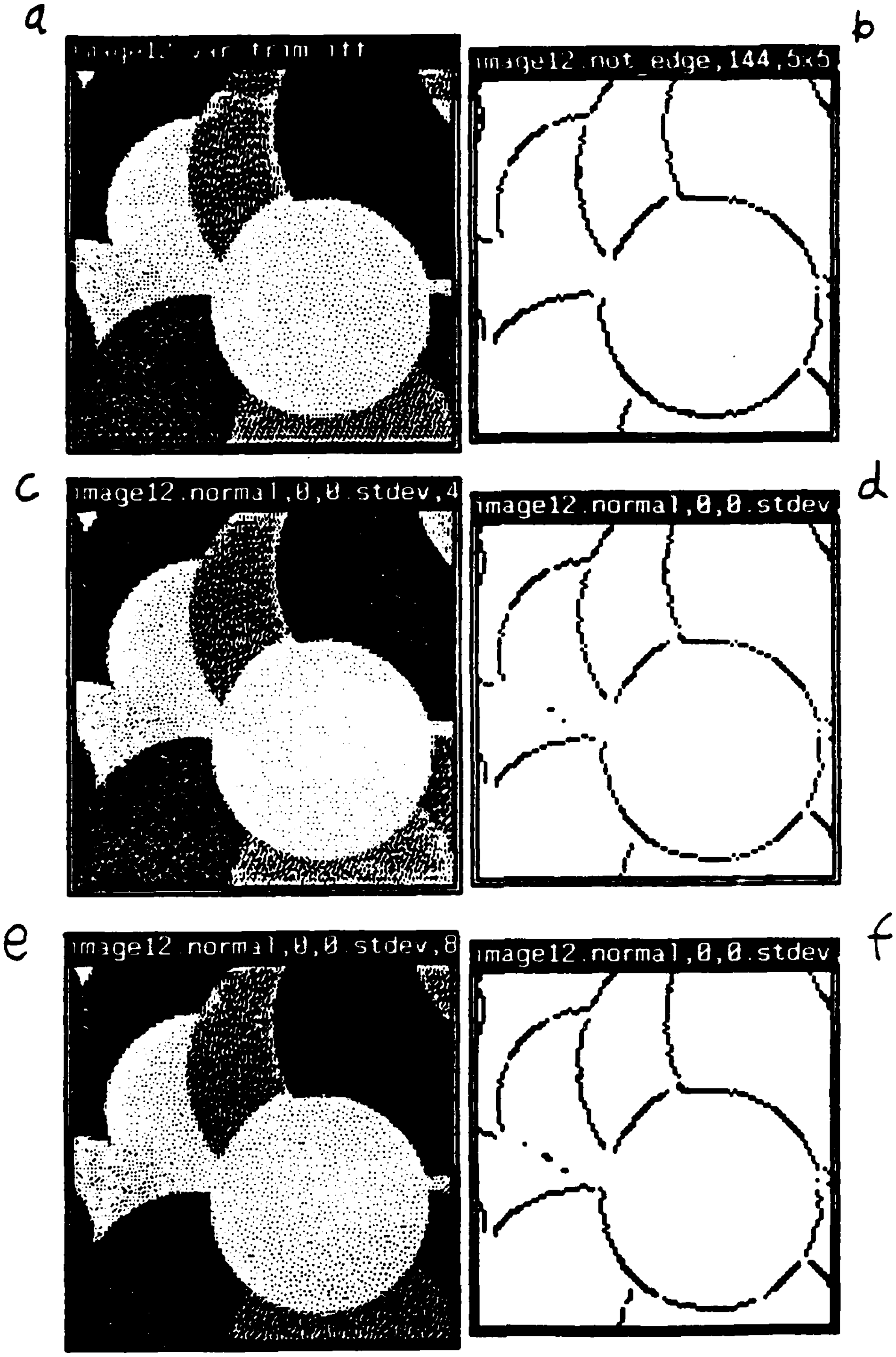
The two circle image (figure 18a) is a particularly good image to test the effect of boundary orientation, curvature and contrast on boundary detection. Figure 19 shows the result of using a 5x5 operator tuned to standard deviation 12 noise on image 18a with standard deviation 12 noise added to it. The images are white at points of greater than 50% probability and black at points less than 50% probability.



a: image with stdev 12 noise | **b: white if there is no edge**
c: white if 0 degree edge | **d: white if 90 degree edge**
e: white if 45 degree edge | **f: white if 135 degree edge**

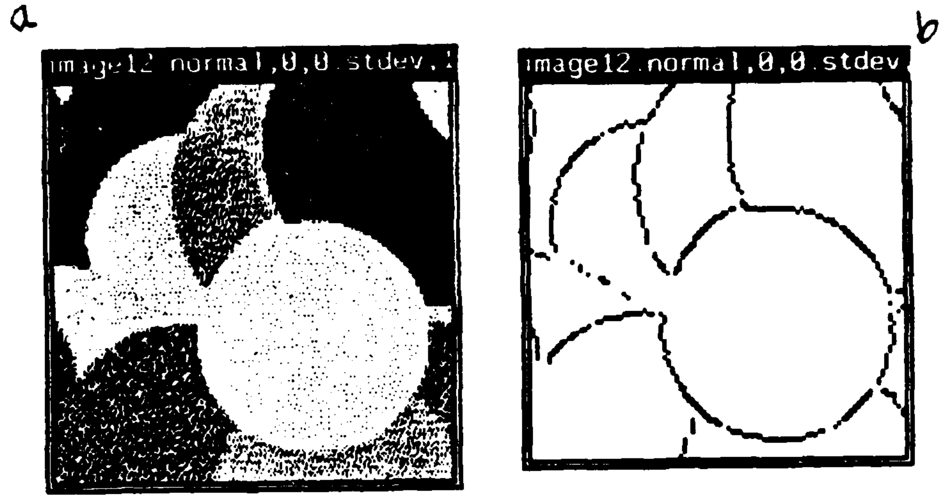
Figure 19: Oriented response for 5x5 operator

In figures 20, 21, and 22 I apply the standard deviation 12 operator to the image 18b with too little, just right and too much noise respectively. The operator output is black when there is greater than 50% probability of a boundary. Note that with too little noise the detector misses boundaries that are there. With too much noise the detector detects boundaries that are not there.



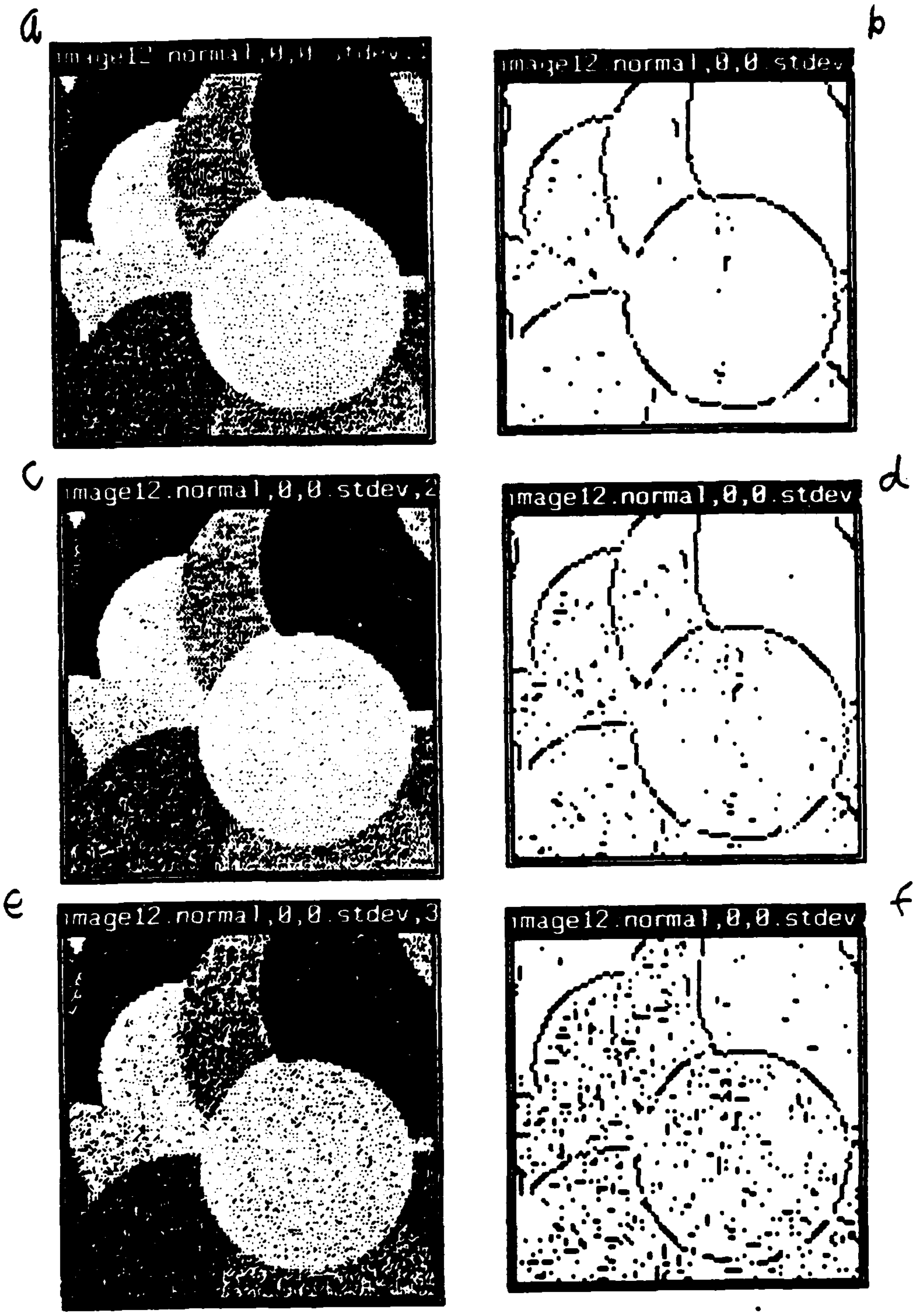
a: image with $\sigma=0$ noise **b: $\sigma=12$ operator on image with $\sigma=0$ noise**
c: image with $\sigma=4$ noise **d: $\sigma=12$ operator on image with $\sigma=4$ noise**
e: image with $\sigma=8$ noise **f: $\sigma=12$ operator on image with $\sigma=8$ noise**

Figure 20: $\sigma=12$ 5x5 operator applied to images with too little ($\sigma<12$) noise



a: image with $\sigma=12$ noise b: $\sigma=12$ operator on image with $\sigma=12$ noise

Figure 21: $\sigma=12$ 5x5 operator applied to image with correct ($\sigma=12$) amount of noise



a: image with $\sigma=16$ noise b: $\sigma=12$ operator on image with $\sigma=16$ noise
c: image with $\sigma=20$ noise d: $\sigma=12$ operator on image with $\sigma=20$ noise
e: image with $\sigma=32$ noise f: $\sigma=12$ operator on image with $\sigma=32$ noise

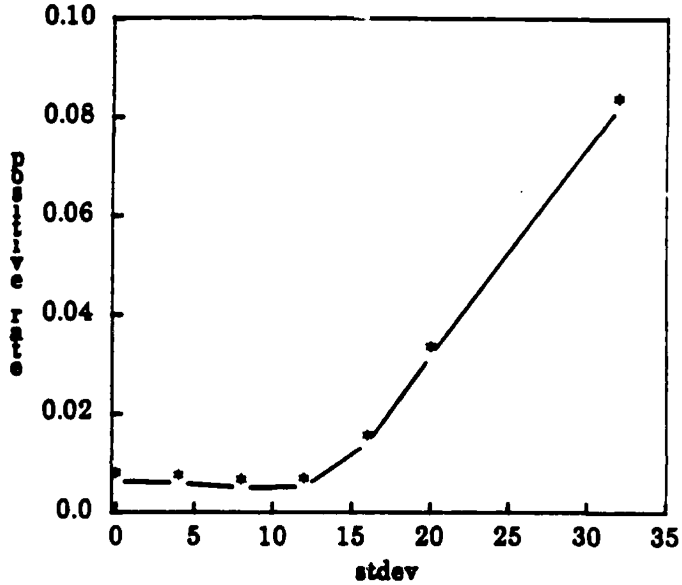
Figure 22: $\sigma=12$ 5x5 operator applied to images with too much ($\sigma>12$) noise

Because these are artificial images I know exactly where the boundaries are. Thus I have written software that counts how many mistakes (false positives and negatives) are made in boundary determination. False negatives are when a boundary that is in the image is missed. False positives are boundaries that are reported where there is no boundary there.

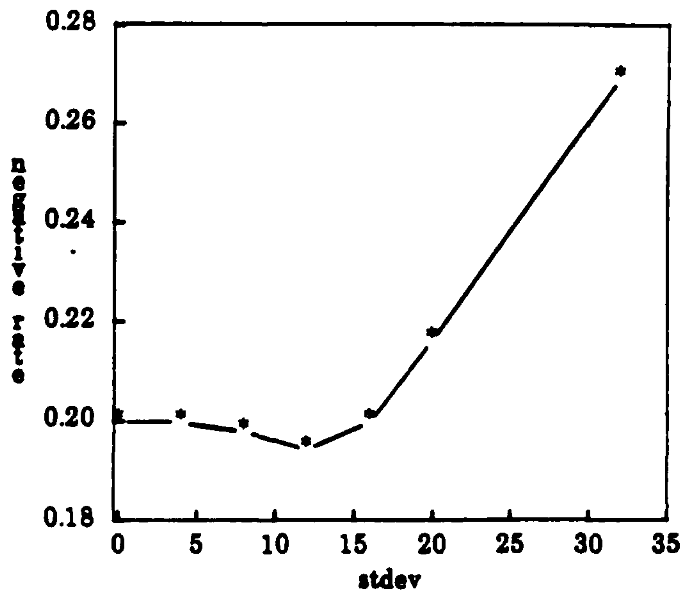
One tricky point is that multiple reports of boundaries are usually considered a bad result [Canny83]. However systems that report a boundary only once will usually have a high false negative rate because they report an edge one pixel off from where it really is. I consider this error to have low enough cost to be ignored. So my software ignores false negatives that are next to reported boundaries in a direction normal to the boundary.

Figure 23 charts the performance of my operator tuned to $\sigma=12$ on the images shown previously. Figure 24 charts the performance of my operator tuned to $\sigma=4$ noise. Figure 25 charts the performance of an operator that is tuned to the same noise level as is contained in the image. Figure 26 superimposes the three graphs of total error rates to show the relationship.

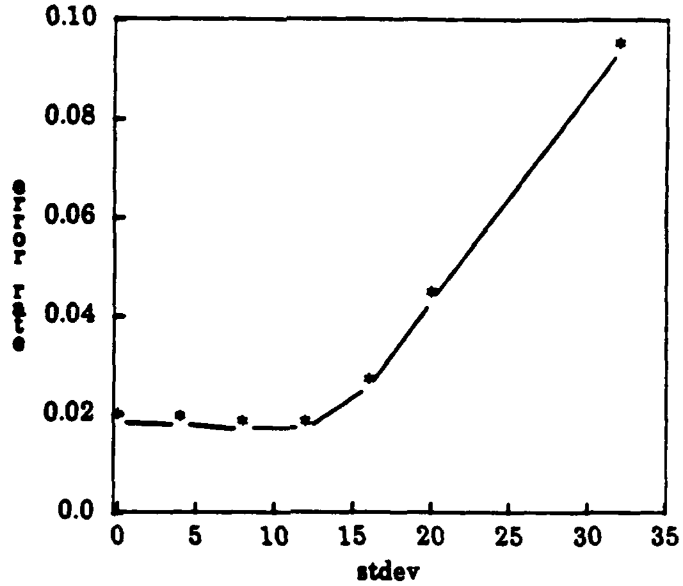
(a)



(b)

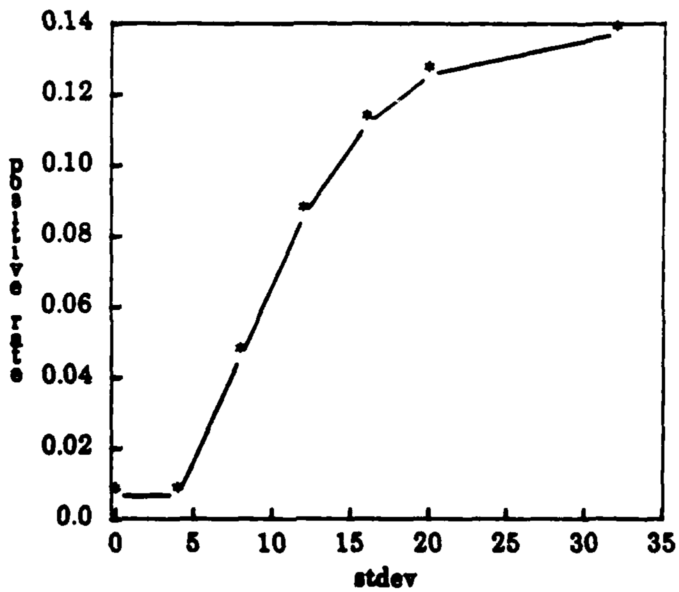


(c)

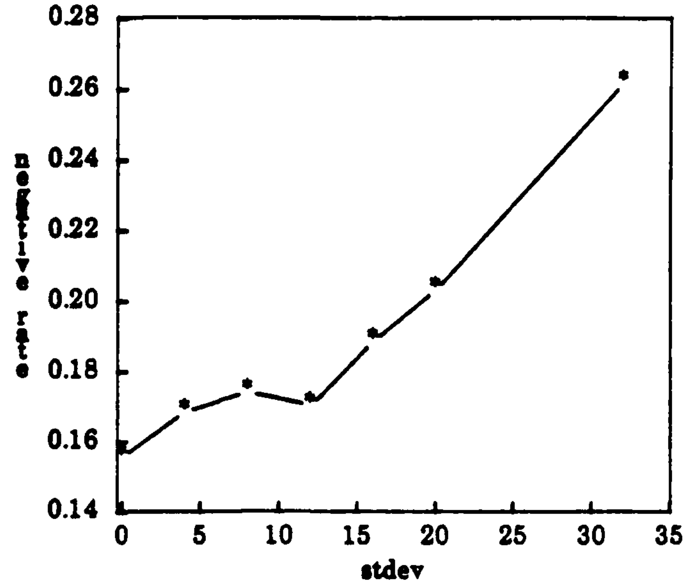


(a) false positive rate vs increasing σ of noise in image
(b) false negative rate vs increasing σ of noise in image
(c) total error rate vs increasing σ of noise in image
Figure 23: Error Rates for the Operator Tuned to $\sigma = 12$ noise

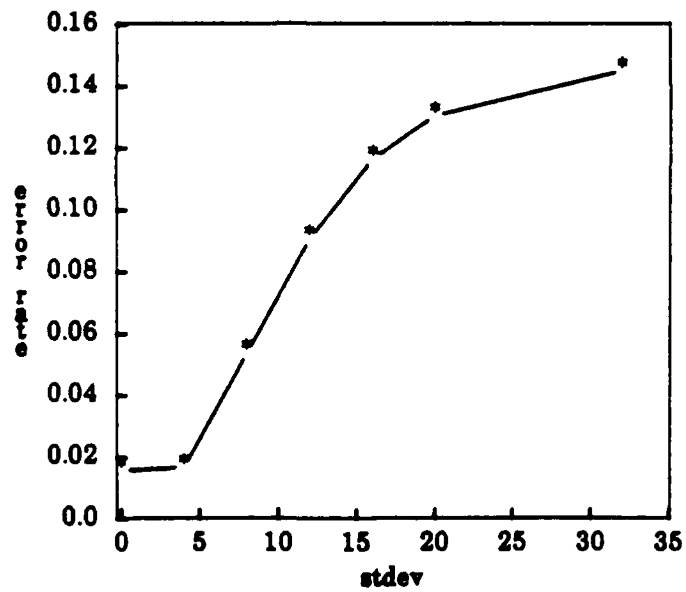
(a)



(b)

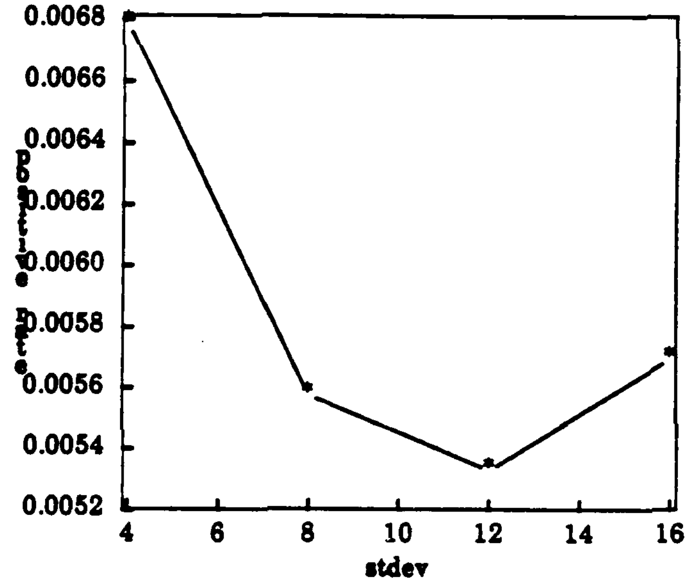


(c)

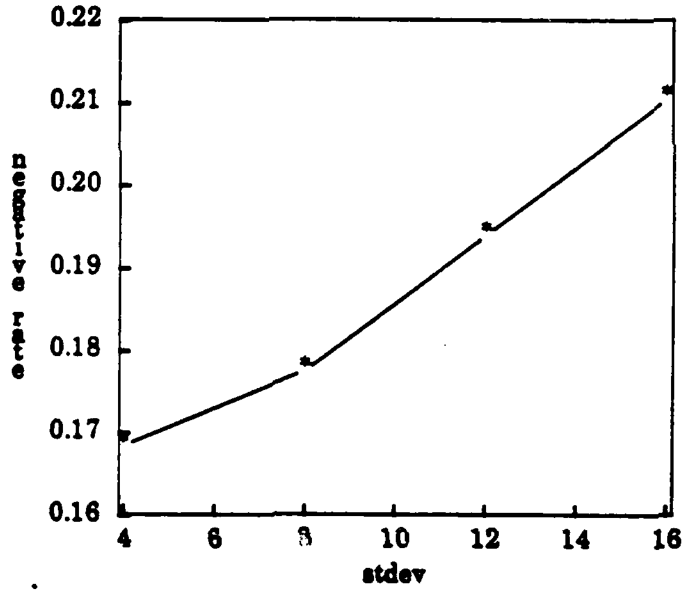


(a) false positive rate vs increasing σ of noise in image
(b) false negative rate vs increasing σ of noise in image
(c) total error rate vs increasing σ of noise in image
Figure 24: Error Rates for the Operator Tuned to $\sigma=4$ noise

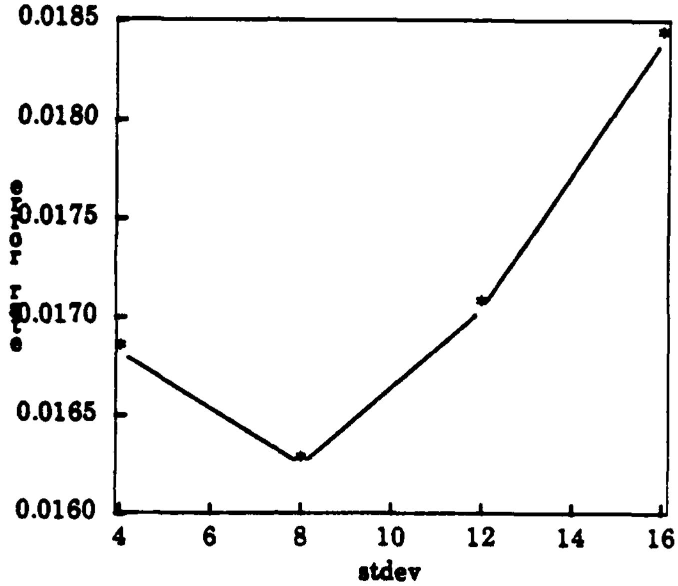
(a)


0.0068
0.0066
0.0064
0.0062
0.0060
0.0058
0.0056
0.0054
0.0052
positive rate
4
6
8
10
12
14
16
stdev

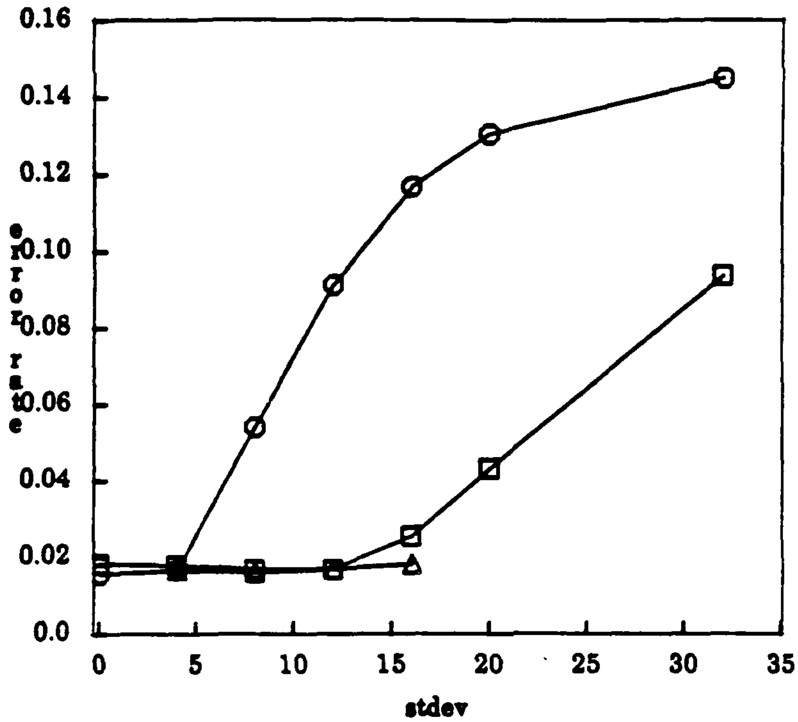
(b)


0.22
0.21
0.20
0.19
0.18
0.17
0.16
negative rate
4
6
8
10
12
14
16
stdev

(c)



(a) false positive rate vs increasing σ of noise in image
(b) false negative rate vs increasing σ of noise in image
(c) total error rate vs increasing σ of noise in image
Figure 25: Error Rates for the Operator Tuned to σ of the noise in the Image



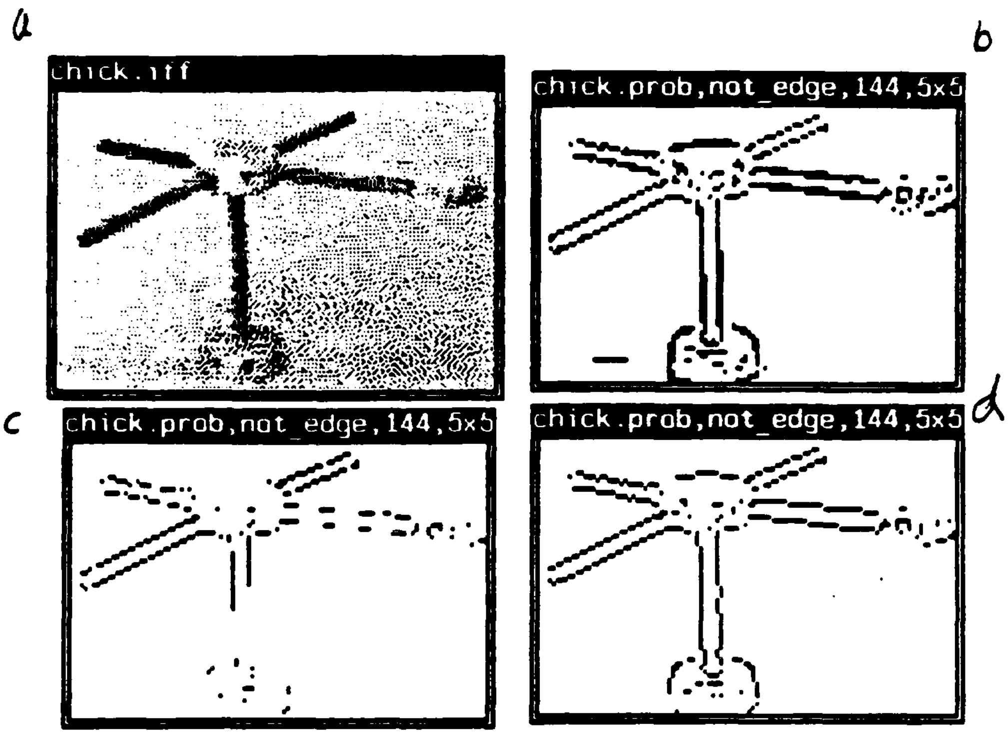
circles: Operator Tuned to $\sigma=12$
squares: Operator Tuned to $\sigma=4$
triangles: Operator Tuned to noise in image

Figure 26: Total error rate for my operators

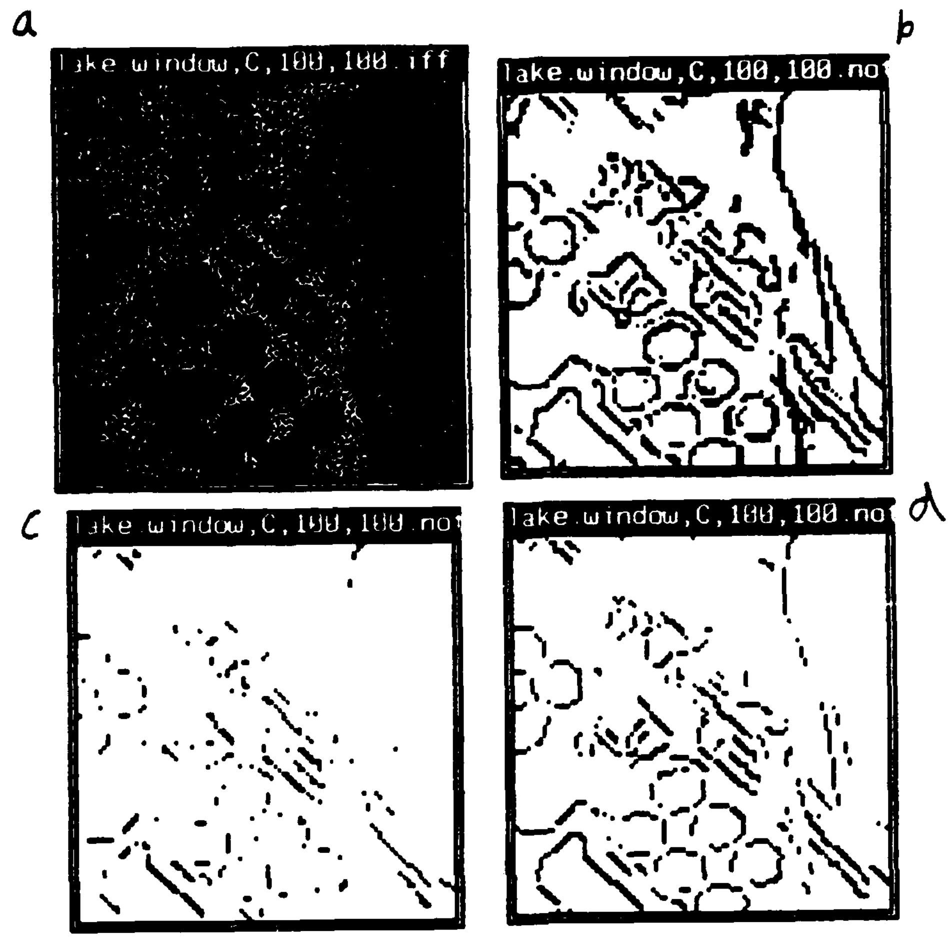
As you can see the tuned operator is always at least as good as the operator that has been developed for stdev 12 or 4 noise.

8.2. Results with Real Images

I have also applied my operator to real images. Here, I use two images, a laboratory image of a Tinkertoy model (figure 27) and an aerial picture (figure 28). I demonstrate the utility of having operators that return probabilities with these results. In both these figures (a) is the image, (b) is the output of my 5x5 stdev 12 operator thresholded at 10% probability, (c) is thresholded at 90% probability and (d) is thresholded at 50% probability. (b) would be used when the cost of missing an edge is high as when the output would be fed to a regularization technique. Note that for case (b) the operator sometimes returns thickened edges. (c) would be used when the cost of missing an edge is low. Hough transform techniques are often developed with that assumption in mind. There is enough information in figure 27c to find the rods of the tinker toy even though all the boundaries are not extant. (d) is what you use when an operator is equally troubled by all errors.



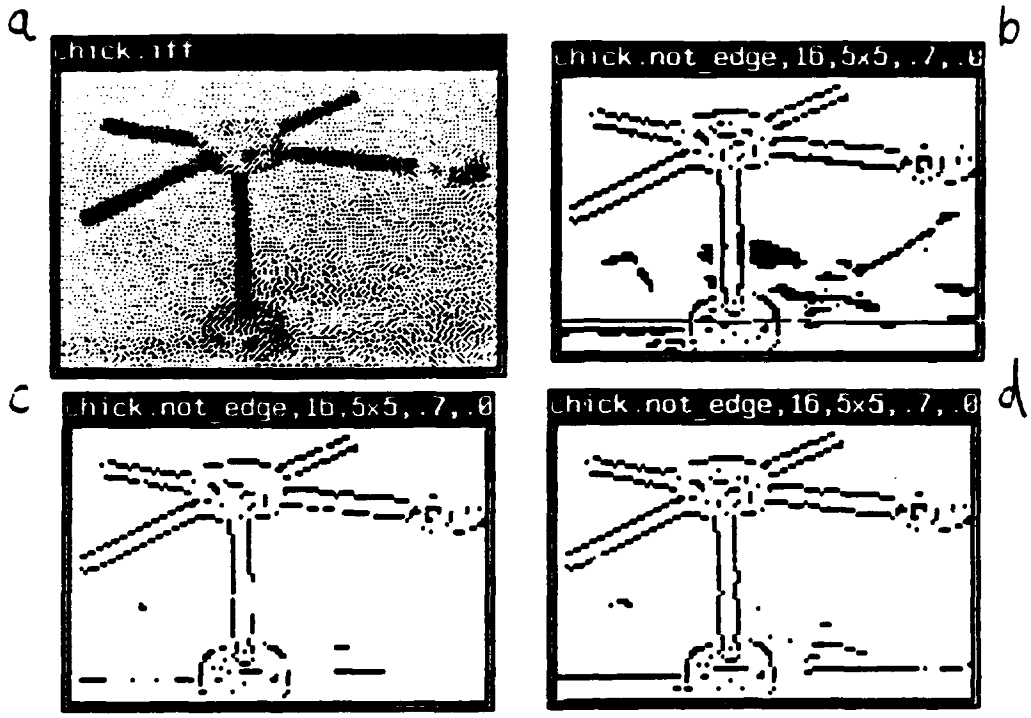
(a) Tinkertoy Image
(b) Output of $\sigma=12$ Operator with threshold at .1
(c) Output of $\sigma=12$ Operator with threshold at .9
(d) Output of $\sigma=12$ Operator with threshold at .5
Figure 27: $\sigma=12$ 5x5 operator applied to tinker toy image



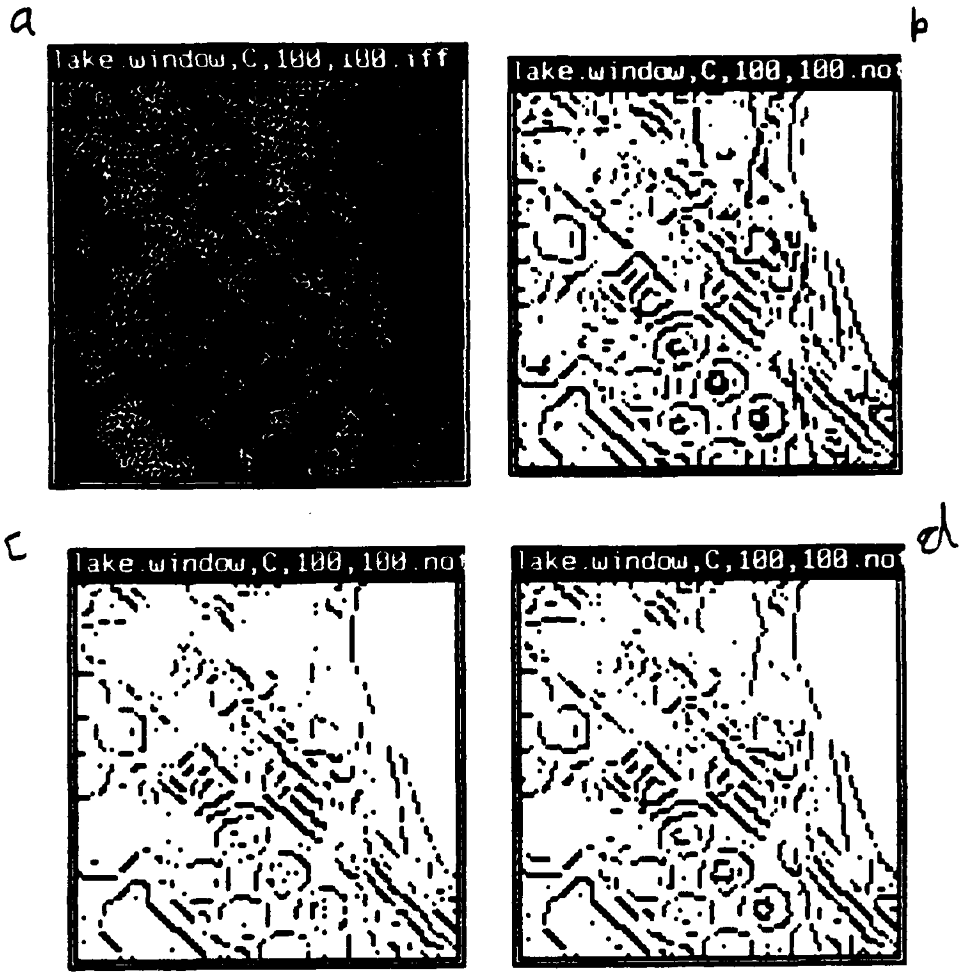
(a) Aerial Image
(b) Output of $\sigma=12$ Operator with threshold at .1
(c) Output of $\sigma=12$ Operator with threshold at .9
(d) Output of $\sigma=12$ Operator with threshold at .5
Figure 28: $\sigma=12$ 5x5 operator applied to aerial image

A particular threshold may result in a most pleasing (to a human observer) ensemble of results (perhaps .5). But this threshold may not be the best threshold for the succeeding application.

One may notice that lowering the threshold seems to increase the number of boundary points in the regions of real boundaries. It is not surprising that the regions that look most like boundaries should be near boundaries. Also the probabilities of boundaries are being reported as lower than they should be in this image. A good reason for this is that the model used to construct the operator shown is not a good model for this image. In particular there is some evidence [Sher87] that the standard deviation of the noise is closer to 4 than 12 in these images. Thus look at the same results for the $\sigma=4$ operator in figures 29 and 30.



(a) Tinkertoy Image
(b) Output of $\sigma=4$ Operator with threshold at .1
(c) Output of $\sigma=4$ Operator with threshold at .9
(d) Output of $\sigma=4$ Operator with threshold at .5
Figure 27: $\sigma=4$ 5x5 operator applied to tinker toy image



(a) Aerial Image
(b) Output of $\sigma=4$ Operator with threshold at .1
(c) Output of $\sigma=4$ Operator with threshold at .9
(d) Output of $\sigma=4$ Operator with threshold at .5
Figure 28: $\sigma=4$ 5x5 operator applied to aerial image

8.3. Comparisons with Established Techniques

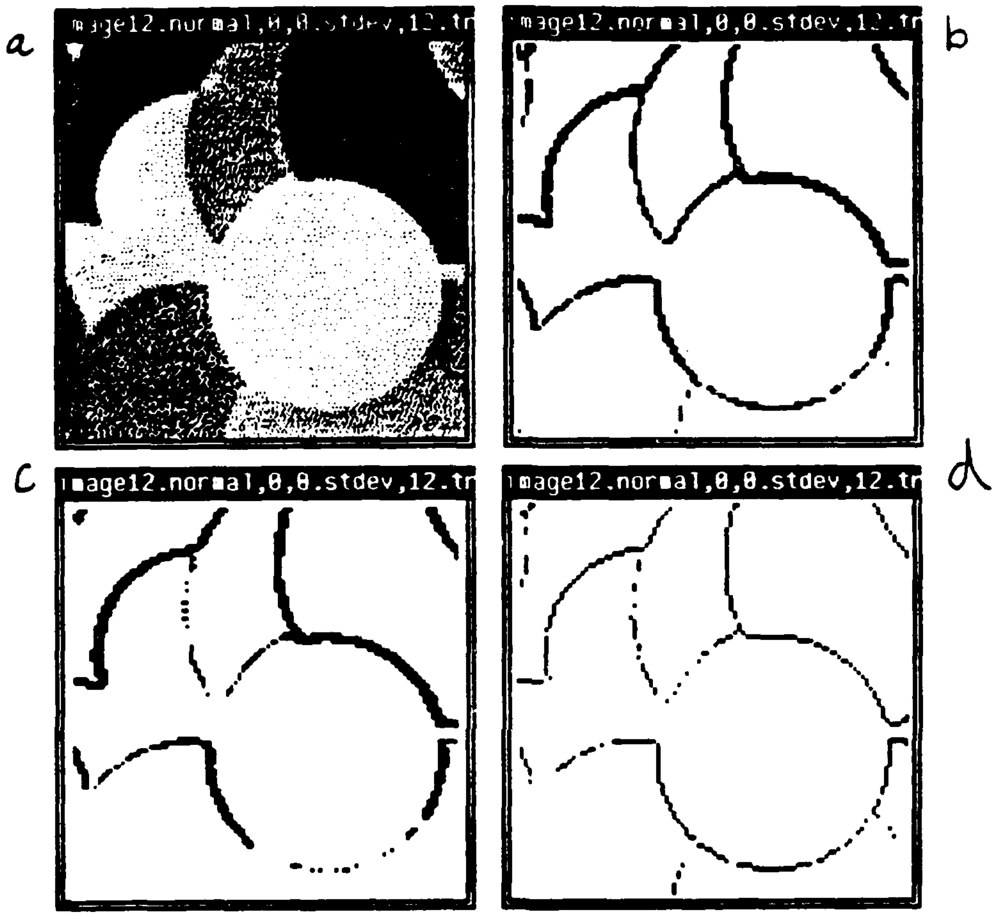
Here, I compare the results I have just presented with established edge detectors. The edge detectors I currently have results for are the Sobel thresholded at 200, the 5 by 5 Kirsch thresholded at 750, and a thinned 3 by 3 Kirsch thresholded at 100. The thresholds for the Sobel and the thinned Kirsch were found to be the ones that minimized the number of errors when applied to the artificial image in figure 18b with standard deviation 12 noise added. The threshold that minimized the errors with the Kirsch was 1175. However at this threshold more than 50% of the boundaries were missed. It was difficult to chart the result of using this operator along with the rest[2] so I used a somewhat lower threshold.

I know these operators are not state of the art. However these results show that tools are available to test this theory against more sophisticated operators. More sophisticated operators

[2]Visual inspection indicated that lowering the threshold would make the result of the operator more appealing. Not that it should matter but ...

such as Canny's and Haralick's will be tested against my operators in the next month or two.

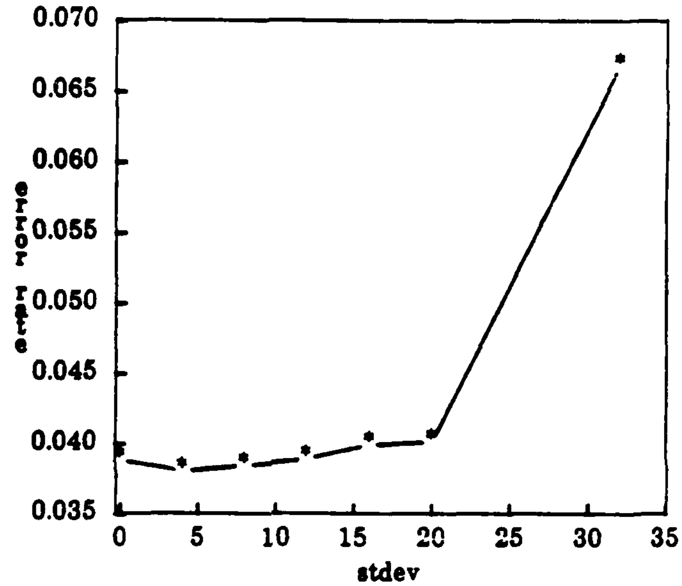
In figure 31 the results of applying the Sobel, Kirsch and thinned Kirsch to my artificial image with stdev 12 noise.



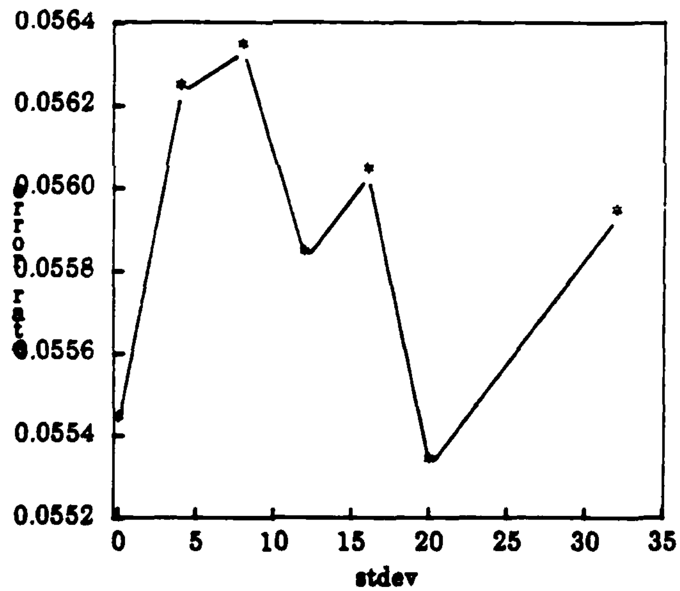
(a) Image with $\sigma = 12$ noise.
(b) Output of the Sobel Optimally Thresholded
(c) Output of the 5x5 Kirsch Optimally Thresholded
(d) Output of the Thinned Kirsch Optimally Thresholded
Figure 31: Application of Established Operators to an Artificial Image

I have measured the error rates for these operators and have charted the total error rates in figure 32.

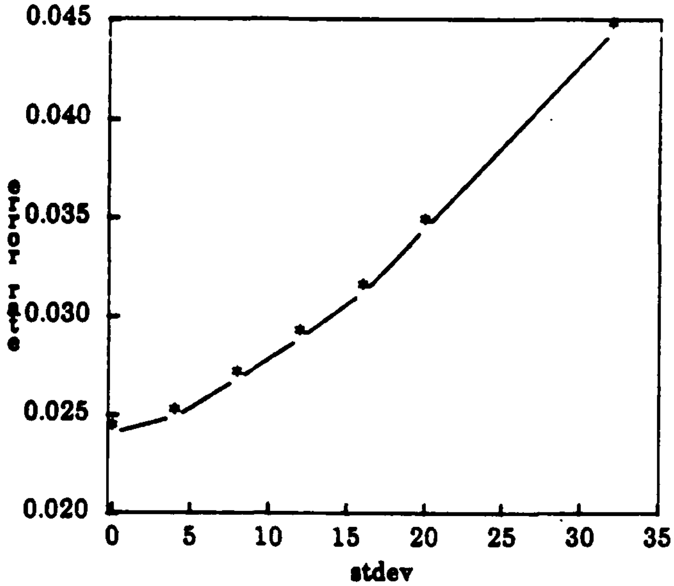
(a)


0.070
0.065
0.060
0.055
0.050
0.045
0.040
0.035
error rate
0
5
10
15
20
25
30
35
stdev

(b)

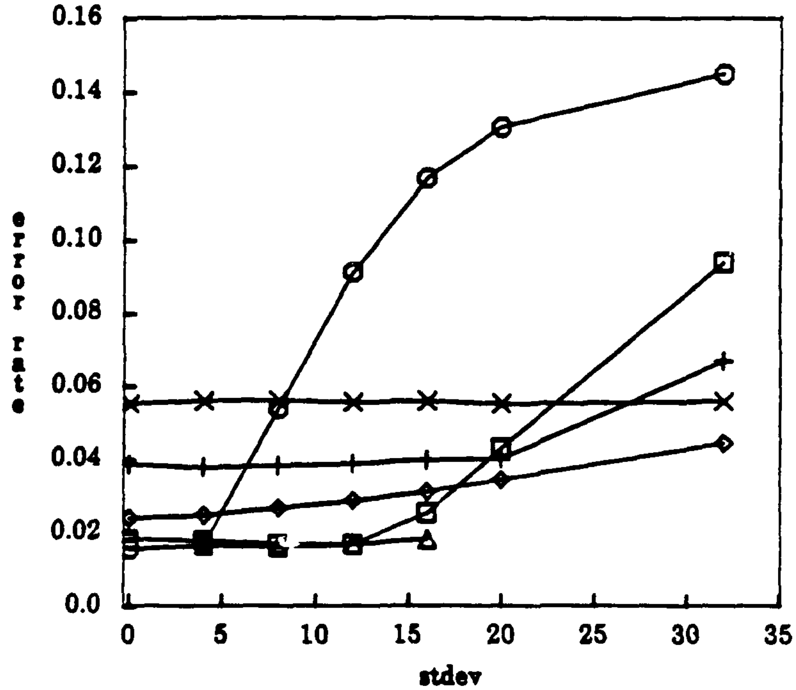

0.0564
0.0562
0.0560
0.0558
0.0556
0.0554
0.0552
error rate
0
5
10
15
20
25
30
35
stdev

(c)



(a) Error rates for the thresholded Sobel
(b) Error rates for the thresholded Kirsch
(c) Error rates for the thinned Kirsch
Figure 32: Charts of Total Error Rates for Established Operators

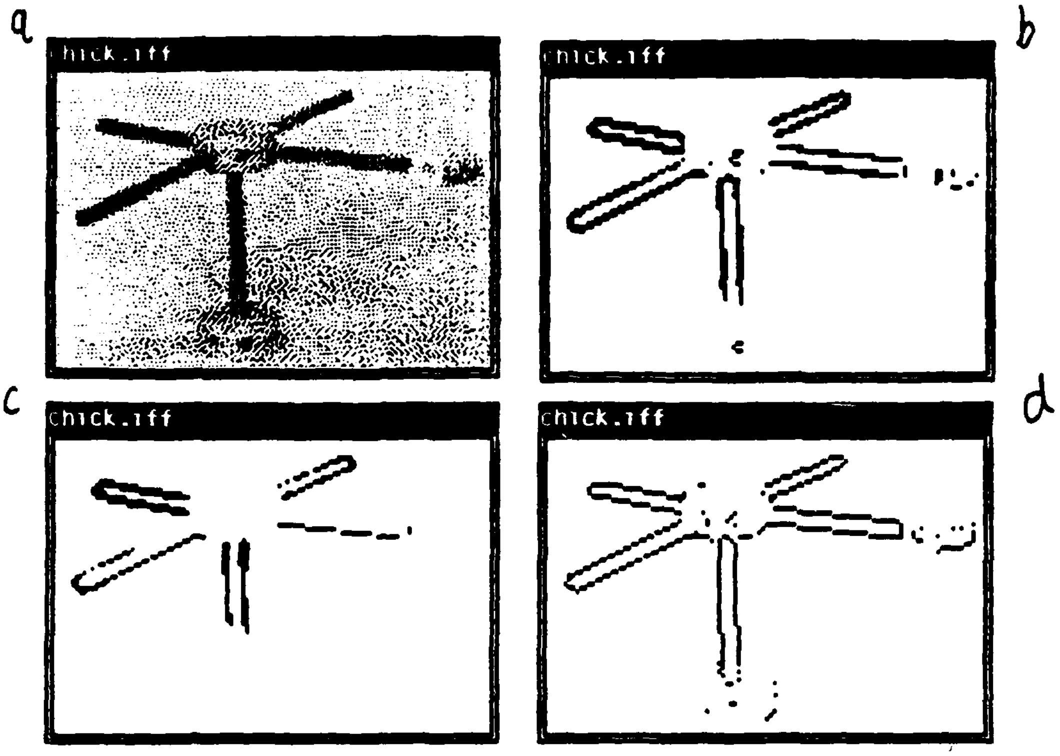
To summarize the results I have a plot that shows the error rates for all the operators I have tested so far (figure 33). In this chart my stdev 12 operator is shown as squares, my tuned operator is shown as circles, the Sobel is shown as triangles, the thresholded Kirsch is shown as crosses and the thinned Kirsch is shown as X's.



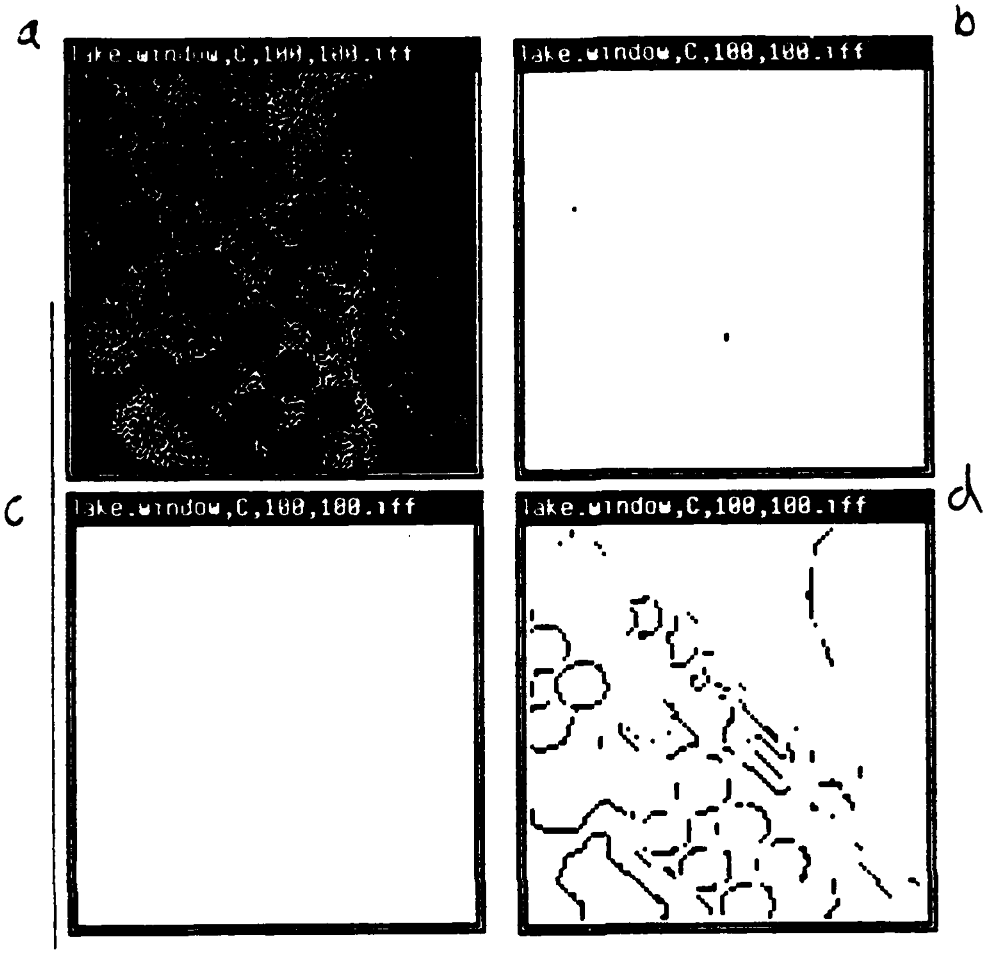
circles:	Tuned $\sigma=12$ operator
squares:	Tuned $\sigma=4$ operator
triangles:	Tuned to noise σ operator
crosses:	Sobel's error rate
X's:	5x5 Kirsch's error rate
diamonds:	Thinned Kirsch's error rate

Figure 33: Error Rates for all Operators

For comparison, I have run the 3 established edge detector on the two real images shown in section 8.2. Figures 34 and 35 show the result of runing these operators with my established operators. Clearly, in these circumstances the most effective established operator for these images is the thinned Kirsch.



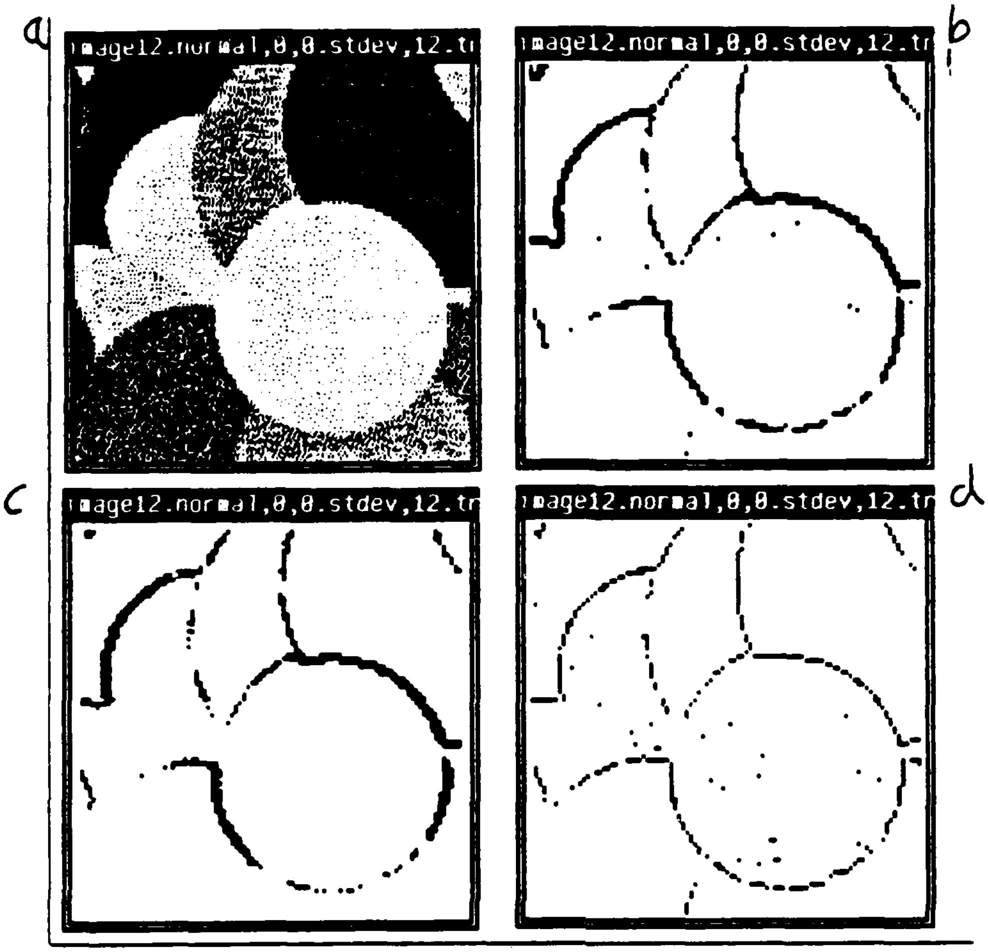
(a) Tinkertoy Image
(b) Output of the Sobel Optimally Thresholded
(c) Output of the 5x5 Kirsch Optimally Thresholded
(d) Output of the Thinned Kirsch Optimally Thresholded
Figure 34: Application of Established Operators to Tinkertoy Image



(a) Aerial Image
(b) Output of the Sobel Optimally Thresholded
(c) Output of the 5x5 Kirsch Optimally Thresholded
(d) Output of the Thinned Kirsch Optimally Thresholded
Figure 35: Application of Established Operators to Aerial Image

One can criticize these comparisons by saying that the image statistics favor my operators, which are robust with dim images. To counter this criticism I have also run the three operators (Sobel, Kirsch and thinned Kirsch) with a preprocessing stage of histogram equalization. Thus all the test images will be rescaled to have the same statistics. Thus when I find the optimal threshold for the standard deviation 12 artificial image it should remain a good threshold for all the tests.

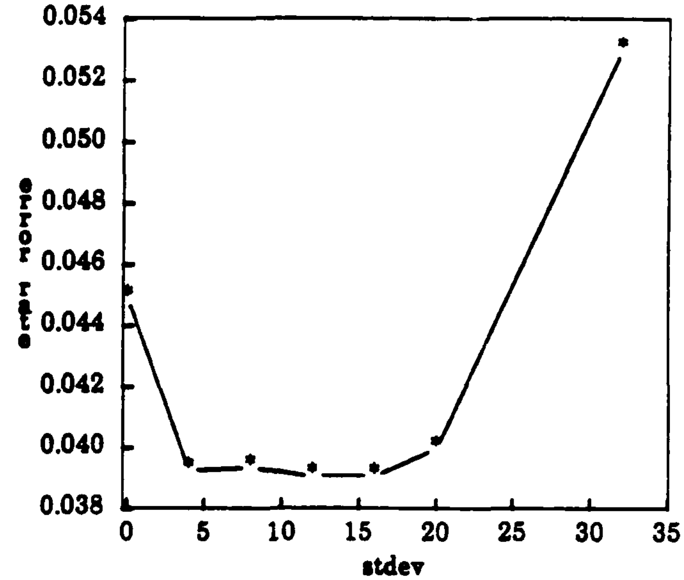
The optimal thresholds happened to be 220 for the Sobel, 750 for the Kirsch (coincidently), and 125 for the thinned Kirsch. The result of using these operators and thresholds on the standard deviation 12 artificial image (figure 18b) is shown in figure 36.



(a) Image with $\sigma=12$ noise.
(b) Output of the Histogram Equalized Sobel Optimally Thresholded
(c) Output of the Histogram Equalized 5x5 Kirsch Optimally Thresholded
(d) Output of the Histogram Equalized Thinned Kirsch Optimally Thresholded
Figure 36: Application of Histogramed Equalized Operators to Artificial Image

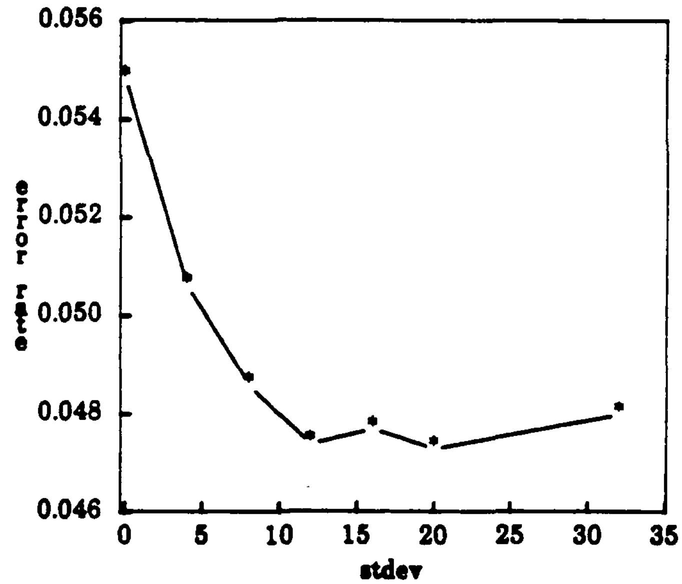
In figure 37 are charts that describe the result of applying these operators to the histogram equalized artificial image (figure 18b).

(a)



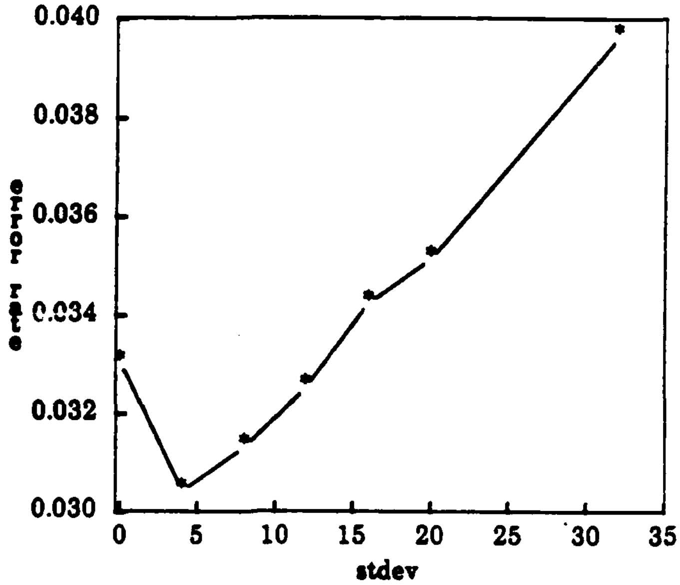
0.054
0.052
0.050
0.048
0.046
0.044
0.042
0.040
0.038
error rate
0
5
10
15
20
25
30
35
stdev

(b)



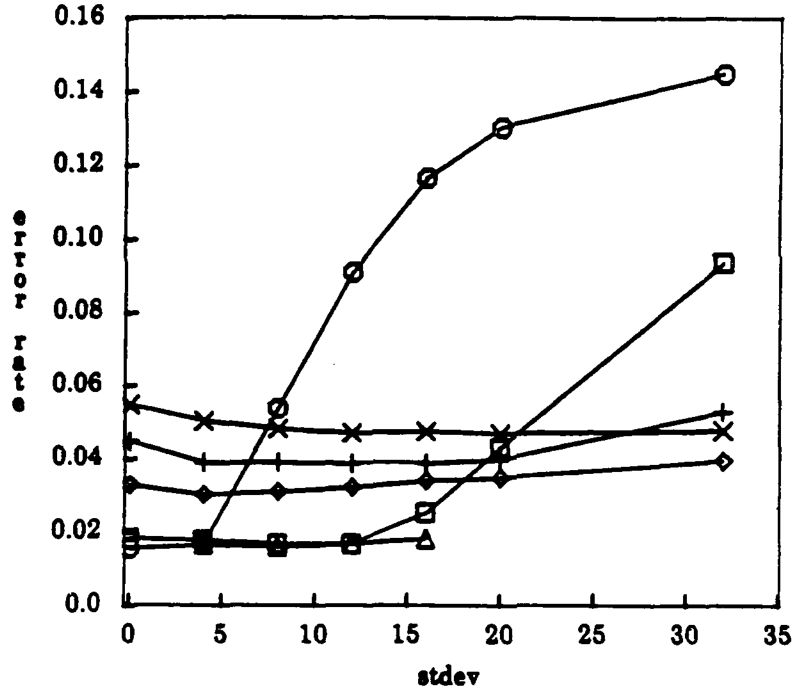
0.056
0.054
0.052
0.050
0.048
0.046
error rate
0
5
10
15
20
25
30
35
stdev

(c)



(a) Total Error rates for the thresholded Sobel on the Histogram Equalized Image
(b) Total Error rates for the thresholded Kirsch on the Histogram Equalized Image
(c) Total Error rates for the thinned Kirsch on the Histogram Equalized Image
Figure 37: Charts of Error Rates for Established Operators

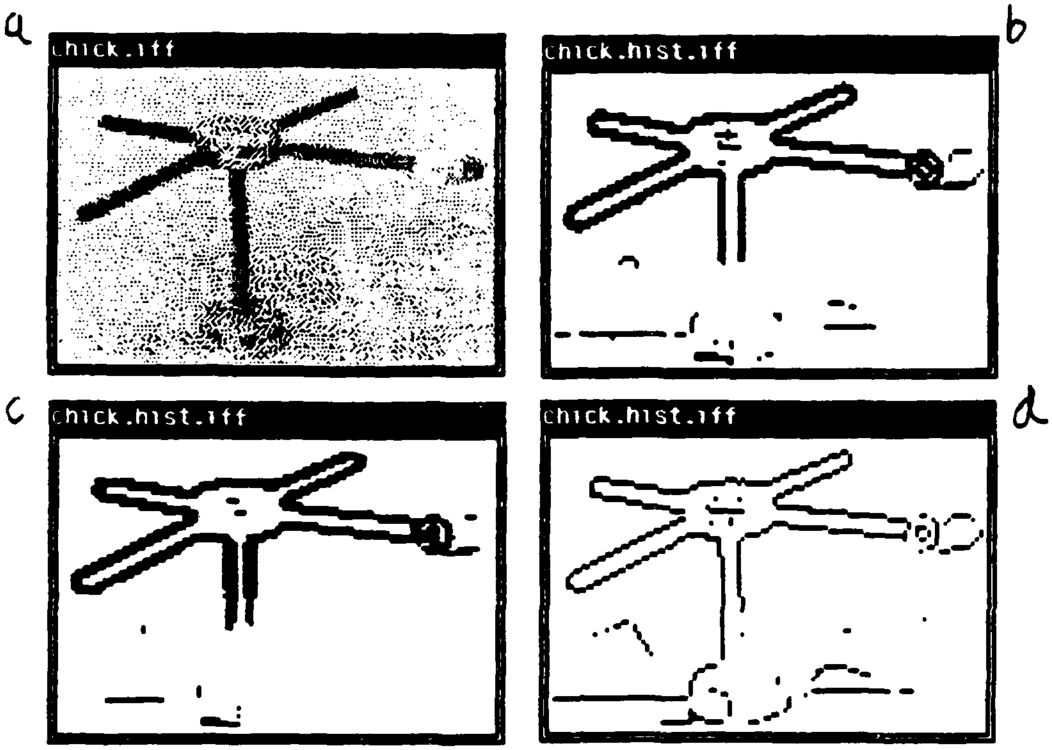
In figure 38 I compare the 3 operators on histogram equalized images to my operators on the original images (my operators do not expect histogram equalization and doing so may confuse them).



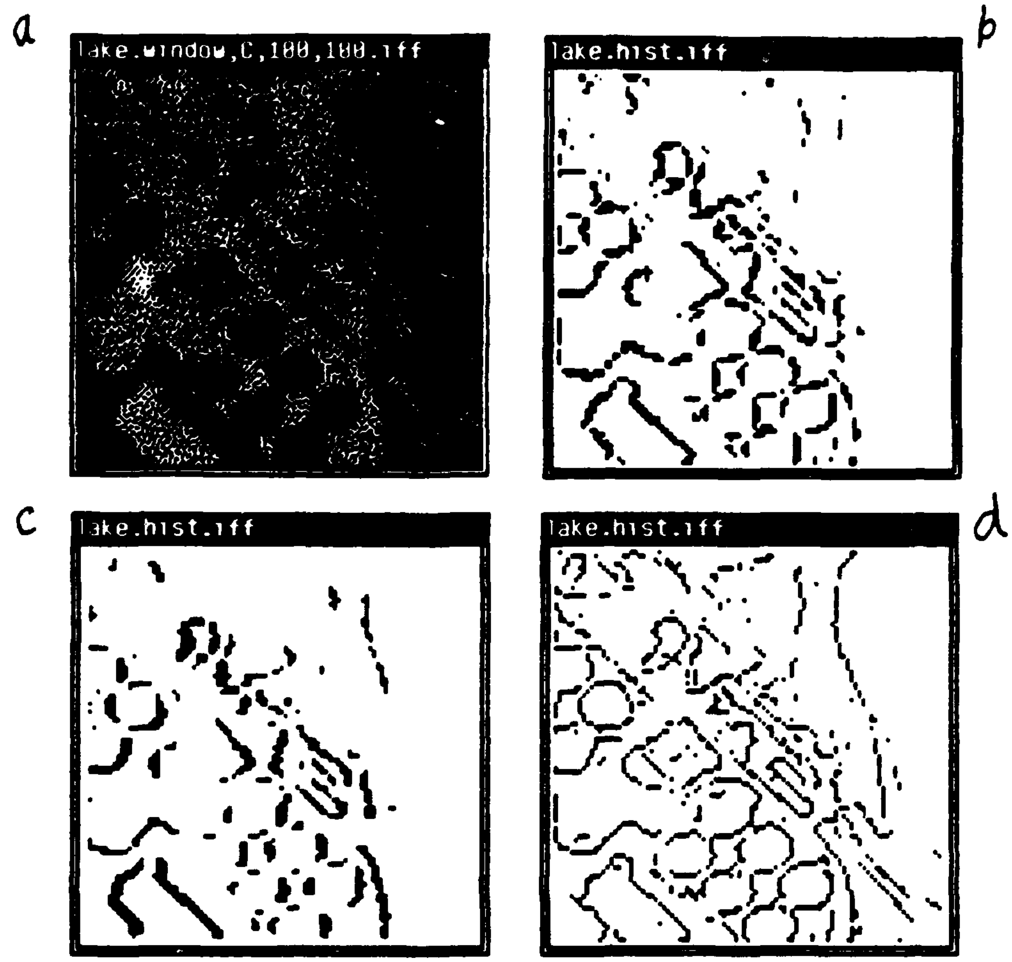
circles:	Tuned $\sigma=12$ operator
squares:	Tuned $\sigma=4$ operator
triangles:	Tuned to noise σ operator
crosses:	Histogram Equalized Sobel's error rate
X's:	Histogram Equalized 5x5 Kirsch's error rate
diamonds:	Histogram Equalized Thinned Kirsch's error rate

Figure 38: Comparison with Established Operators Applied to Histogram Equalized Image

Since the artificial images already had the full range of graylevels histogram equalization did not help the established operators much. However the advantage of histogram equalization is shown clearly when the operators are applied to real images in figures 39, and 40.



(a) Tinkertoy Image
(b) Output of the Histogram Equalized Sobel Optimally Thresholded
(c) Output of the Histogram Equalized 5x5 Kirsch Optimally Thresholded
(d) Output of the Histogram Equalized Thinned Kirsch Optimally Thresholded
Figure 39: Application of Established Operators to Tinkertoy Image



(a) Aerial Image
(b) Output of the Histogram Equalized Sobel Optimally Thresholded
(c) Output of the Histogram Equalized 5x5 Kirsch Optimally Thresholded
(d) Output of the Histogram Equalized Thinned Kirsch Optimally Thresholded
Figure 40: Application of Established Operators to Aerial Image

9. Previous Work

Edge detection is the established vision task that bears most closely on the boundary detection problem I describe here. Edge detection has been one of the earliest and most important tasks attempted by computer vision systems. Usually edge detection is described as a problem in image reconstruction.

Edge detection is often characterized as discovering the contrast in a region of the ideal image when there is an boundary between two constant intensity regions of the ideal image. Since I am not motivated by image reconstruction this task is not of particular interest for me. However often edge detection algorithms are used for boundary point detection. The idea is to accept as boundaries the pixels whose windows the detector considers to have high contrast.

The first work on edge detection was by Roberts who developed the Roberts edge operator to detect boundaries and corners of blocks. It was a simple convolution operator probably inspired by convolution based pattern matching. Since Roberts edge detection has been worked on by a large number of vision workers. Some of the operators were worked out in a somewhat ad hoc manner

as the Roberts was. The "best" and most common example of such operators is the Sobel edge operator

Many have worked on "optimal operators" where some model of edges is presented and the "best" function that fits a specified functional form. The definition of "best" and the functional form varies. Almost everyone who takes this approach limits the functional forms of edge operators to convolutions.

Hueckel [Hueckel71] considers convolutions over a disc on the image. He models edges as step edges with linear boundaries occurring at random places in the disk. His functions are limited to look at only certain specific Bessel coordinates (of an integral Fourier transform) that he has determined are useful for edge detection. He takes into account somewhat the possibility of two edges in the region. In a later paper [Hueckel73] Hueckel considers edges that are two parallel step edges a few pixels apart. He analyzes such edges the same way he analyzed the previous kind of edges.

At MIT starting with Marr [Marr82] there has been concentration on zero crossing based edge detection. The edge detectors they use are to locate edges at zero crossings of a Laplacian of a Gaussian. [Torre86] [Lunscher86b] [Lunscher86a] describe how such an edge detector is an approximation to a spline based operator that has maximal output at edges (compared to elsewhere). Such an operator also has been shown always to create connected boundaries.

Canny [Canny83] has examined the issue of convolution based edge detection more closely. In particular he studied the goals of edge detection. He considered an edge detector to be good if it reported strongly when there was an edge there and *did not report* when there was no edge. He also wanted a detector that only reported once for an edge. He found that these constraints conflict when one is limited to convolution based edge detectors (such behavior arises naturally for the boundary point detectors in this paper). His primary work on this topic was with a 1 dimensional step edge model. He derived a convolution operator that was similar to a 0 crossing operator. He also discusses how to extend the operators defined for one dimensional images to two dimensional images, and when oriented operators are desirable. His operators, applied to real images, usually appear to do a good job of finding the boundaries. In this paper I derive boundary point detectors for step edges but do not constrain the functional form of the edge detector. Thus the edge detectors based on this should have performance at least as good as Canny's detector.

Nalwa [Nalwa84] used a more sophisticated model where he assumed that regions in the intensity image fit (at least locally) surfaces that are planar, cubic or tanh type. He tested whether a surface fit a window on the image and if not he tried to fit various boundaries between surfaces. He ordered the tests to be of increasing computational complexity. His operators, applied to real images, usually appear to do a good job of finding the boundaries. The work in this paper handles models of this form and derives optimal operators.

Another approach to edge detection is to simulate parts of the human early visual system. Zero crossing operators were originally motivated by this argument since it was found that there were cells in the human early visual system that compute zero crossings at various frequencies [Marr82]. Other work that seriously studies the human early visual system was by Fleet [Fleet84] on the spatio-temporal properties of center-surrounds operators in the early human visual system. My work is not concerned with the structure of the early human visual system since its goals are to perform a task best as possible rather than as human-like as possible. However I can draw inspiration from the human visual system since it has been highly optimized to its goals by

evolution and hence an optimal detection system may be similar to that of the human eye (or animal eye for that matter).

Haralick has taken a similar approach to mine for the problem of edge detection [Haralick86a]. The differences between his approach and mine are that he models the image as a surface rather than as a function of a scene, and his operators generate decisions about edges rather than probabilities of edges [Haralick84]. However he has told me that his theory can be used to generate likelihoods that can be used with the techniques presented here [Haralick86b]. The relationship between his facet model and my template based models is currently under investigation.

10. Conclusion

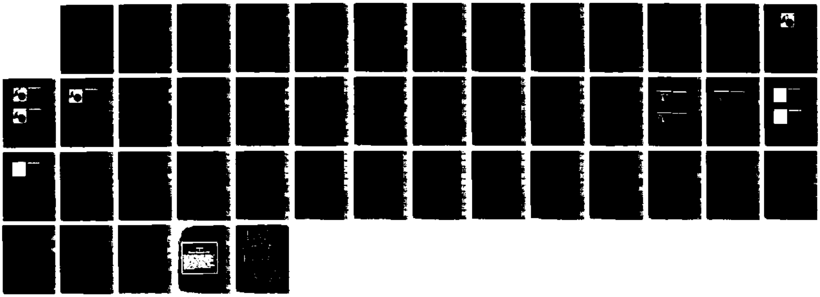
I have demonstrated an operator that fulfills the desiderata in section 1. It has flexible output that can be used by many operators because it returns probabilities. It works on gray scale input. Because the operator is based on windows it does work proportional to the size of the image to calculate boundary probabilities. By constructing templates to represent a boundary shifted less than a pixel one can have subpixel precision with work proportional to that precision. A parameter of the algorithms I describe is the expected distribution of illuminances. Another is the standard deviation of and correlation in the noise.

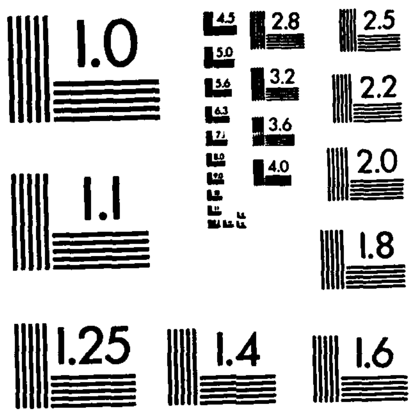
Results were reported from using a 5 by 5 operator developed from this theory in section 8. I have applied this detector to artificial and real images. In section 8.3 I have compared my detectors to the established detectors, Sobel, Kirsch, and thinned Kirsch. In the next few weeks results from using a 7 by 7 and 9 by 9 operator will be available. Also comparisons will be done between these detectors and more advanced edge detectors such as Canny's [Canny83], and Haralick's [Haralick84].

In a companion report I describe an evidence combination theory that is applied to operators that return likelihoods that allows me to combine robustly the output of several different operators on the same data [Sher87]. Soon there will be results from using the likelihoods as input to a Markov random field based system [Chou87].

AD-A198 597 NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL 3/3
REPORT 1986 VOLUME 8..(U) SYRACUSE UNIV NY C M BROWN
JUN 88 RADC-TR-88-11-VOL-8-PT-B F30602-85-C-0008

UNCLASSIFIED F/G 12/9 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

References

[Andrews77]
H. C. Andrews and B. R. Hunt, *Digital Image Restoration*, 8-26 , Prentice-Hall, INC.., Englewood Cliffs, New Jersey 07632, 1977

[Berger80] J. O. Berger, *Statistical Decision Theory*, 110-112 , Springer-Verlag., New York Heidelberg Berlin, 1980

[Binford81]
T. O. Binford, Inferring Surfaces from Images, *Artificial Intelligence 17*,1-3 (August 1981), 205-244, North-Holland Publishing Company.

[Boult6?] T. E. Boult and J. R. Kender, *On Visual Surface Reconstruction Using Sparse Depth Data*, Department of Computer Science, Columbia University., 1986?

[Brown82] C. M. Brown, Bias and Noise in the Hough Transform 1: theory, 105, Department of Computer Science, University of Rochester, June 1982.

[Canny83] J. F. Canny, Finding Edges and Lines in Images, 720, MIT Artificial Intelligence Laboratory, June 1983.

[Chou87] P. Chou and D. Sher, (Markov Random Fields for Information Fusion Based Segmentation) ?, To be Published in 87.

[Fleet84] D. J. Fleet, The Early Processing of Spatio - Temporal Visual Information, 84-7, University of Toronto, Research in Biological and COmputational Vision, September 1984.

[Geman84]
S. Geman and D. Geman, Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images, *PAMI 6*,6 (Novermber 1984), 721-741, IEEE.

[Haralick84]
R. M. Haralick, Digital Step Edges from Zero Crossing of Second Directional Derivatives, *PAMI 6*,1 (January 1984), 58-68, IEEE.

[Haralick86a]
R. M. Haralick, The Facet Approach to Gradient Edge Detection, *Tutorial 1 Facet Model Image Processing (CVPR)*, May 1986.

[Haralick86b]
R. Haralick, Personal Communication, June 1986.

[Hueckel71]
M. H. Hueckel, An Operator Which Locates Edges in Digitized Pictures, *Journal of the Association for Computing Machinery 18*,1 (January 1971), 113-125, ACM.

[Hueckel73]
M. H. Hueckel, A local Visual Operator Which Recognizes Edges and Lines, *J. ACM 20*,4 (October 1973), 634-647, ACM.

[Lunscher86a]
W. H. H. J. Lunscher and M. P. Beddoes, Optimal Edge Detector Design II: Coefficient Quantization, *Pattern Analysis and Machine Intelligence 8,2* (March 1986), 178-187, IEEE.

[Lunscher86b]
W. H. H. J. Lunscher and M. P. Beddoes, Optimal Edge Detector Design I: Parameter Selection and Noise Effects, *Pattern Analysis and Machine Intelligence 8,2* (March 1986), 164-177, IEEE.

[Marr82] D. Marr, *Vision*, W. H. Freeman and Company., New York, 1982

[Marroquin85]
J. L. Marroquin, Probabilistic Solution of Inverse Problems, Tech. Rep. 860, MIT Artificial Intelligence Laboratory, September 1985.

[Nalwa84] V. S. Nalwa, On Detecting Edges, *Proceedings: Image Understanding Workshop*, October 1984, 157-164.

[Sher85] D. B. Sher, Evidence Combination for Vision using Likelihood Generators, *Proceedings: Image Understanding Workshop (DARPA)*, Miami, Florida, December 1985, 255-270. Sponsored by: Information Processing Techniques Office Defence Advanced Research Projects Agency.

[Sher86] D. Sher, Optimal Likelihood Detectors for Boundary Detection Under Gaussian Additive Noise, *IEEE Conference on Computer Vision and Pattern Recognition*, Miami, Florida, June 1986.

[Sher87] D. B. Sher, Evidence Combination Based on Likelihood Generators, TR192, University of Rochester Computer Science Department, Milan, Italy, January 1987. Submitted in shorter form to IJCAI.

[Torre86] V. Torre and T. A. Poggio, On Edge Detection, *Pattern Analysis and MAchine Intelligence 8,2* (March 1986), 147-163, IEEE.

Appendix B-7

Evidence Combination Using Likelihood Generators

David Sher
Computer Science Department
The University of Rochester
Rochester, New York 14627

January 1987
TR 192

Appendix B-7

Evidence Combination Using Likelihood Generators

David Sher
Computer Science Department
The University of Rochester
Rochester, New York 14627

January 1987
TR192

Abstract

Here, I address the problem of combining output of several detectors for the same feature of an image. I show that if the detectors return likelihoods I can robustly combine their outputs. The combination has the advantages that:

- The confidences of the operators in their own reports are taken into account. Hence if an operator is confident about the situation and the others are not then the reports of the confident operator dominates the decision process.
- *A priori* confidences in the different operators can be taken into account.
- The work to combine 'N' operators is linear in 'N'.

This theory has been applied to the problem of boundary detection. Results from these tests are presented here.

This work would have been impossible without the advice and argumentation of such people as Paul Chou and Mike Swain (who has made suggestions from the beginning) and of course my advisor Chris Brown, and who could forget Jerry Feldman. This work was supported by the Defense Advanced Research Projects Agency U. S. Army Engineering Topographic Labs under grant number DACA76-85-C-0001. Also this work was supported by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB, DC 20332, under Contract No. F30602-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC).

1. Introduction

Often in computer vision one has a task to do such as deriving the boundaries of objects in an image or deriving the surface orientation of objects in an image. Often one also has a variety of techniques to do this task. For boundary detection there are a variety of techniques from classical edge detection literature [Ballard82] and the image segmentation literature e.g. [Ohlander79]. For determining surface orientation there are techniques that derive surface orientation from intensities [Horn70] and texture [Ikeuchi80] [Aloimonos85]. These techniques make certain assumptions about the structure of the scene that produced the data. Such techniques are only reliable when their assumptions are met. Here I show that if several algorithms return likelihoods I can derive from them the correct likelihood when at least one of the algorithms' assumptions are met Thus I derive an algorithm that works well when any of the individual algorithms works well.

The mathematics here were derived independently but are similar to the treatment in [Good50]. and [Good83], using different notation. To understand my results first one must understand the meaning of likelihood.

2. Likelihoods

In this paper I call the assumptions that an algorithm makes about the world a *model*. Most models for computer vision problems describe how configurations in the real world generate observed data. Because imaging projects away information, the models do not explicitly state how to derive the configuration of the real world from the sensor data. As a result, graphics problems are considerably easier than vision problems. Programs can generate realistic images that no program can analyze.

Let O be the observed data, f a feature of the scene whose existence we are trying to determine (like a boundary between two pixels) and M a model. Many computer vision problems can be reduced to finding the probability of the feature given the model and the data, $P(f|O\&M)$. However most models for computer vision instead make it easy to compute $P(O|f\&M)$. I call $P(O|f\&M)$ (inspired by the statistical literature) the *likelihood* of f given observed data O under M. As an example assume f is "the image has a constant intensity before noise". M says that the image has a normally distributed uncorrelated (between pixels) number added to each pixel (the noise). Calculating $P(O|M\&f)$ is straight-forward (a function of the mean and variance of O).

A theorem of probability theory, *Bayes' law*, shows how to derive conditional probabilities for features from likelihoods and prior probabilities. Bayes' law is shown in equation 1.

$$P(f|O\&M)=\frac{P(O|f\&M)P(f|M)}{P(O|f\&M)P(f|M)+P(O|\sim f\&M)P(\sim f|M)} \tag{1}$$

f is the feature for which we have likelihoods. M is the domain model we are using. $P(O|f\&M)$ is the likelihood of f under M and $P(f|M)$ is the probability under M of f

For features that can take on several discrete mutually exclusive labels (rather than just true and false) such as surface orientation (which can be a pair of angles to the nearest degree or "not applicable" (at boundaries)) a more complex form of Bayes' law shown in equation 2 yields conditional probabilities from likelihoods and priors.

$$P(l|O\&M) = \frac{P(O|l\&M)P(l|M)}{\sum_{l' \in L(f)} P(O|l'\&M)P(l'|M)} \tag{2}$$

l is a label for feature f and $L(f)$ is the set of all possible labels for feature f.

Another important use for explicit likelihoods is for use in Markov random fields. Markov random fields describe complex priors that can capture important information. Several people have applied Markov random fields to vision problems [Geman84]. Likelihoods can be used in a Markov random field formulation to derive estimates of boundary positions [Marroquin85b] [Chou87]. In [Sher86] and [Sher87] I discuss algorithms for determining likelihoods of boundaries.

Let us call an algorithm that generates likelihoods a *likelihood generator*. Different models lead to different likelihood generators. The difference between two likelihood generators' models can be a single constant (such as the assumed standard deviation of the noise) or the two likelihood generators' models may not resemble each other in the slightest.

Consider likelihood generators L_1 and L_2 with models M_1 and M_2 and assume they both determine probability distributions for the same feature. L_1 can be considered to return the likelihood of a label l for feature f given observed data O and the domain model M_1. Thus L_1 calculates $P(O|f=l\&M_1)$. Also L_2 calculates $P(O|f=l\&M_2)$. A useful combination of L_1 and L_2 is the likelihood detector that returns the likelihoods for the case where M_1 or M_2 is true. Also the prior confidences one has in M_1 and M_2 should be taken into account.

This paper studies deriving $P(O|f=1\&(M_1 \vee M_2))$. Note that if I can derive rules for combining likelihoods for two different models then by applying the combination rules N times, N likelihoods are combined. Thus all that is needed is combination rules for two models.

3. Combining Likelihoods From Different Models

To combine likelihoods derived under M_1 and M_2 an examination of the structure and interaction of the two models is necessary. M_1 and M_2 must have the same definition for the feature being detected. If the feature is defined differently for M_1 and M_2 then M_1 and M_2 are about different events, and the likelihoods can not be combined with the techniques developed in this section.

Thus the likelihood generated by an occlusion boundary detector can not be combined with the likelihood generated by a detector for boundaries within the image of an object (such as corners internal to the image). A detector of the likelihood of heads on a coin flip can not be combined with a detector of the likelihood of rain outside using this theory. (However easy it may be using standard probability theory.)

If the labeling of a feature f implies a labeling for another feature g then in theory one can combine a f detector with a g detector by using the g detector that is implied by the f detector. As an example a region grower could be combined with a boundary detector since the position of the regions implies the positions of the boundaries.

3.1. Combining Two Likelihoods

The formula for combining the likelihoods generated under M_1 and M_2 requires prior knowledge. Necessary are the prior probabilities $P(M_1)$ and $P(M_2)$ that the domain models M_1 and M_2 are correct as well as $P(M_1\&M_2)$. Often $P(M_1\&M_2) = 0$. When this occurs the two models contradict each other. I call two such models *disjoint* because both can not describe the situation

simultaneously. If M_1 is a model with noise of standard deviation $4\pm\varepsilon$ and M_2 is a model with noise of standard deviation $8\pm\varepsilon$ then their assumptions contradict and $P(M_1 \& M_2) = 0$.

Prior probabilities for the feature labels under each model ($P(f=l|M_1)$ and $P(f=l|M_2)$) are necessary. If $P(M_1 \& M_2) \neq 0$ then the prior probability of the feature label under the conjunction of M_1 and M_2 ($P(f=l|M_1 \& M_2)$) and the output of a likelihood generator for the conjunction of the two models ($P(O|f=l \& (M_1 \& M_2))$) are needed. If I have this prior information I can derive $P(O|f=l \& (M_1 \vee M_2))$.

If I were to combine another model, M_3, with this combination I need the priors $P(M_3)$, $P(f|M_3)$, $P(M_3 \& (M_1 \vee M_2))$ and $P(f|M_3 \& (M_1 \vee M_2))$. To add on another model I need another 4 priors. Thus the number of prior probabilities to combine n models is linear in n.

Thus all that is left is to derive the combination rule for likelihood generators given this prior information. The derivation starts by applying the definition of conditional probability in equation 3.

$$P(O|f=l \& (M_1 \vee M_2)) = \frac{P(O \& f=l \& (M_1 \vee M_2))}{P(f=l \& (M_1 \vee M_2))} \tag{3}$$

The formula for probability of a disjunction is applied to the numerator and denominator in equation 4.

$$P(O|f=l \& (M_1 \vee M_2)) = \frac{P(O \& f=l \& M_1)+P(O \& f=l \& M_2)-P(O \& f=l \& M_1 \& M_2)}{P(f=l \& M_1)+P(f=l \& M_2)-P(f=l \& M_1 \& M_2)} \tag{4}$$

In equation 5 the definition of conditional probability is applied again to the terms of the numerator and the denominator.

$$P(O|f=l \& (M_1 \vee M_2)) = \frac{\left[\begin{array}{c} P(O|f=l \& M_1)P(f=l|M_1)P(M_1) \\ + \\ P(O|f=l \& M_2)P(f=l|M_2)P(M_2) \\ - \\ P(O|f=l \& M_1 \& M_2)P(f=l|M_1 \& M_1)P(M_1 \& M_2) \end{array}\right]}{P(f=l|M_1)P(M_1)+P(f=l|M_2)P(M_2)-P(f=l|M_1 \& M_2)P(M_1 \& M_2)} \tag{5}$$

Different assumptions allow different simplifications to be applied to the rule in equation 5. If the two models are disjoint equation 5 reduces to equation 6.

$$P(O|f=l \& (M_1 \vee M_2)) = \frac{\left[\begin{array}{c} P(O|f=l \& M_1)P(f=l|M_1)P(M_1) \\ + \\ P(O|f=l \& M_2)P(f=l|M_2)P(M_2) \end{array}\right]}{P(f=l|M_1)P(M_1)+P(f=l|M_2)P(M_2)} \tag{6}$$

Another assumption that simplifies things considerably is the assumption that prior probabilities for all feature labelings in all the models and combinations thereof are the same. I call this assumption *constancy of priors*. When constancy of priors is assumed $P(f=l|M_1) = P(f=l|M_2) = P(f=l|M_1 \& M_2)$. Making this assumption reduces the number of priors that need to be determined. Since determining prior probabilities from a model is sometimes a difficult task the constancy of priors is a useful simplification. With constancy of priors equation 5 reduces to equation 7.

$$P(O|f=l\&(M_1 \vee M_2)) = \frac{\begin{bmatrix} P(O|f=l\&M_1)P(M_1) \\ + \\ P(O|f=l\&M_2)P(M_2) \\ - \\ P(O|f=l\&M_1\&M_2)P(M_1\&M_2) \end{bmatrix}}{P(M_1)+P(M_2)-P(M_1\&M_2)} \tag{7}$$

Equation 6 with constancy of priors reduces to equation 8.

$$P(O|f=l\&(M_1 \vee M_2)) = \frac{\begin{bmatrix} P(O|f=l\&M_1)P(M_1) \\ + \\ P(O|f=l\&M_2)P(M_2) \end{bmatrix}}{P(M_1)+P(M_2)} \tag{8}$$

Thus equation 8 describes the likelihood combination rule with disjoint models and constancy of priors.

3.2. Understanding the Likelihood Combination Rule

The easiest incarnation of the likelihood combination rule to understand is the rule for combining likelihoods from disjoint models given constancy of priors across models (equation 8). Here the combined likelihood is the weighted average of the likelihoods from the individual models weighted by the probabilities of the models applying. (The combined likelihood is the likelihood given the disjunction of the models).

If models M_1 and M_2 are considered equally probable and the likelihoods returned by M_1's detector are considerably larger than those of M_2's detector then the probabilities determined from the combination of M_1 and M_2 are close to those determined from M_1. Thus a model with large likelihoods determines the probabilities. To illustrate this principle consider an example.

Assume that a coin has been flipped $n+1$ times. The results of flipping it has been reported for the first n times. The task is to determine the probability of heads having been the result of the $n+1^{st}$ flip. Consider the results of each coin flip independent. Let M_1 be the coin being fair so that the probability of heads and tails is equal. Let M_2 be that the coin is biased with the probability of heads is π and tails $1-\pi$ with π being a random choice with equal probability between p and $1-p$. Hence the coin is biased towards heads or tails with equal probability but the bias is consistent between coin tosses. The probability of heads remains the same for all coin tosses in both models. M_1 and M_2 are disjoint (the coin is either fair or it isn't but not both) and the prior probability of a flip being heads or tail is the same for both, .5.

Under M_1 the probability of each of the possible flips of $n+1$ coins is 2^{-n-1}. Under M_2 the probability of $n+1$ flips of coins with h heads and $t=n+1-h$ tails is:

$$\tfrac{1}{2}p^h(1-p)^t + \tfrac{1}{2}p^t(1-p)^h$$

Let $n=2$ and $p=.9$. Assume the first two flips are both heads. Let H be "the third flip was heads" and T be "the third flip was tails." The likelihood of H given the observed data is the probability of all 3 flips being heads divided by the probability of the third flip being heads. The likelihood of T given the observed data is the probability of the first 2 being heads and the 3rd tails divided by the probability of the third flip being tails.

Under M_1 the probability of all 3 flips being heads is 0.125 and the probability of a flip being heads is 0.5 thus the likelihood of H is 0.25. The likelihood of T is 0.25 by the same reasoning.

Applying Bayes' law to get the probability of H under M_1 one derives a probability of .5 .

Under M_2 the probability of all 3 flips being heads is 0.365 and the probability of a flip being heads is 0.5. Thus the likelihood of H is 0.73. Under M_2 the probability of the first two being heads and the third being tails is 0.045 and the probability of a flip being tails is 0.5. Thus the likelihood of T is 0.09. Applying Bayes' law under M_2 a probability of H being 0.89 is derived.

If M_1 and M_2 are considered equally probable then the combination of the likelihoods from the two models is the average of the two likelihoods. Thus the likelihood of H for this combination is 0.49 and the likelihood of T is 0.17 (likelihoods don't have to sum to 1). Bayes' law combines these probabilities to get 0.74 for the 3[rd] flip to be heads.

The table in figure 1 describes combining various M_2's with different values of p with M_1 for the different combinations with $n=4$

Observed Coin Flips	Combined with M_1 or just M_2	Likelihood of H $p=.6$	$p=.9$	Likelihood of T $p=.6$	$p=.9$	Probability of H $p=.6$	$p=.9$
HHHH	Just M_2	0.088	0.5905	0.0672	0.0657	0.567	0.8999
	Combined	0.07525	0.3265	0.06485	0.0641	0.537	0.8359
HHHT	Just M_2	0.0672	0.0657	0.0576	0.0081	0.5385	0.8902
	Combined	0.C8485	0.0641	0.06005	0.0353	0.5192	0.6449
HHTT	Just M_2	0.0576	0.0081	0.0575	0.0081	0.5	0.5
	Combined	0.06005	0.0353	0.06	0.0353	0.5	0.5
HTTT	Just M_2	0.0576	0.0081	0.0672	0.0657	0.4615	0.1098
	Combined	0.06005	0.0353	0.06485	0.0641	0.4808	0.3551
TTTT	Just M_2	0.0672	0.0657	0.088	0.5905	0.433	0.1001
	Combined	0.06485	0.0641	0.07525	0.3265	0.4629	0.1641

Figure 1: Result of likelihood combination Rule

Look at the probabilities with $p=.9$ and the observed data is HHHH. For this case the observed data fits M_2 much better than M_1 and the probability from combining M_1 and M_2 is close to the probability resulting from using just M_2, .9. If we had a longer run of heads the probability of future heads would approach exactly M_2's prediction, .9. On the other hand if we had a long run of equal numbers of heads and tails the probability of future heads would quickly approach the prediction of M_1, .5. When the observed data is HHHT the observed data fits M_1 about as well as M_2 and the resulting probability is near the average of .5 predicted by M_1 and 0.8902 predicted by M_2. Thus when the observed data is a good fit for a particular model (like M_2) the probabilities predicted by the combination is close to the probabilities predicted by the fitted model. If two models fit about equally then the result is an average of the probabilities [1].

4. When No Model Applies

Given a set of likelihood generators and their models, using the evidence combination described in section 3 we can get the likelihood for the feature labelings given that at least one model applies. Thus if we have likelihoods of a boundary given models with the noise standard

[1]However the feature that the decision theory predicts is not the average of the features predicted under the two different models in general.

deviations near to 4, 8 and 16 in them we can derive the likelihood of a given the noise standard deviation is near to 4 or 8 or 16 (no matter which). Thus we can derive the probability distribution over feature labelings given that at least one of our models applies. However what we are trying to derive is the *physical probability distribution* over the feature labelings. This is the probability distribution over feature labels given the observed data (estimated by the long run frequencies over the feature labels given the observed data). The problem is that there may be a case where none of the models assumptions is true. In the Venn diagram of figure 2 each set represents the set of situations where a model's assumptions are true. The area marked NO MODEL is the set of situations where all the models fail.

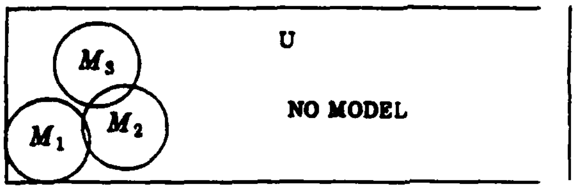


Figure 2: Venn Diagram of Models

What should the likelihood of a feature label be if no model applies? To answer this question I examine the companion question of what should the probability of a feature label be if no model applies. Assume a prior probability for the label is available. If a posterior probability is different from a prior probability for the feature then information has been added to get the posterior. (Only information can justify changing from the prior.) Since having no model means intuitively having no information then the posterior should be the same as the prior. If and only if the likelihoods of all feature labels are equal, the posterior probability is the same as the prior. Hence the likelihoods of the feature labels should be equal for any particular piece of observed data. In this section I assume a prior proability distribution is a available over feature labels. If no such distribution is available an uninformative prior can be constructed [Frieden85].

To constrain the problem further, consider whether any piece of observed data should be more probable than any other when no model applies. It seems unreasonable that one could conclude that some observations are more probable than others without any model of how those observations were produced. Hence all the likelihoods should be equal. This constraint is sufficient to determine the likelihoods when no model applies. I think that this solution minimizes cross entropy with the prior (since it returns the prior) [Johnson85].

To derive the physical probability distribution over feature labels, the "no model" likelihoods should be combined with the likelihoods derived for the models. The probability of each of the models and their combinations must have been available to use the combination rules from section 3. Hence the probability that one or more of the models applies is known. The probability of no model is 1 minus that probability. The conjunction of some model applying and no model applying has 0 probability. Hence combination rule 6 can be applied to derive the likelihoods under any conditions from the likelihoods for any model applying.

As example consider the problem of seeing HHHH and trying to derive the probability of a fifth head given the equally likely choices that the coin is fair or is biased to .9 (biased either for heads or tails with equal probability). The combined likelihood of H is 0.3265 (from figure 1). The combined likelihood of T is 0.0641. As an example, assume that the probabilities that the assumptions of M_1 were true was 0.4 and similar for M_2. Then 0.4 of the time we feel the coin is

fair, 0.4 of the time we feel it has been biased by 0.9, and 0.2 of the time we have no model about what happened. The likelihood of HHHH under "NO MODEL" is .0625 regardless of H or T (Since the likelihood of all 4 coin flip events are equal and must sum to 1). Combining the "NO MODEL" likelihoods with likelihoods of 0.2737 for H and 0.06378 for T (see figure 1), the probability of H from applying Bayes' law to these likelihoods is 0.811. This probability is somewhat nearer to .5 than the probability of 0.8359 derived without taking the possibility of all the models failing into account.

Taking the possibility of all models failing lends certain good properties to the system. Probabilities of 0 or 1 become impossible without priors of 0 or 1. Thus the system is denied total certainty. Numbers near 0 or 1 cause singularities in the equations under finite precision arithmetic. Total certainty represents a willingness to ignore all further evidence. I find that property undesirable in a system. Denying the system total certainty also results in the property that the system must have all probability distribution over feature labels between ϵ and $1-\epsilon$ for an ϵ proportional to the probability that no model applies. Thus there is a limit to how certain our system is about any feature labeling in our uncertain world.

5. Results

I have applied this evidence combination to the boundary detection likelihood generators described in [Sher87]. Here I prove my claims that the evidence combination theory allows me to take a set of algorithms that are effective but not robust and derive an algorithm that is robust. The output of such an algorithm is almost as good as the best of its constituents (the algorithms that are combined).

5.1. Artificial Images

Artificial images were used to test the algorithms described in section 3 quantitatively. I used as a source of likelihoods the routines described in [Sher87]. Because the positions of the boundaries in an artificial image are known one can accurately measure false positive and negative rates for different operators. Also one can construct artificial images to precise specifications. The artificial images I use is an image composed of overlapping circles with constant intensity and aliasing at the boundaries shown in figure 3.

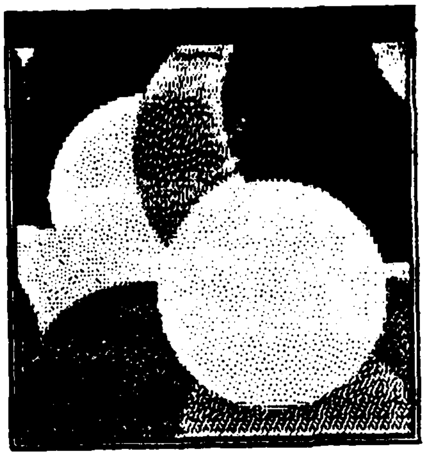
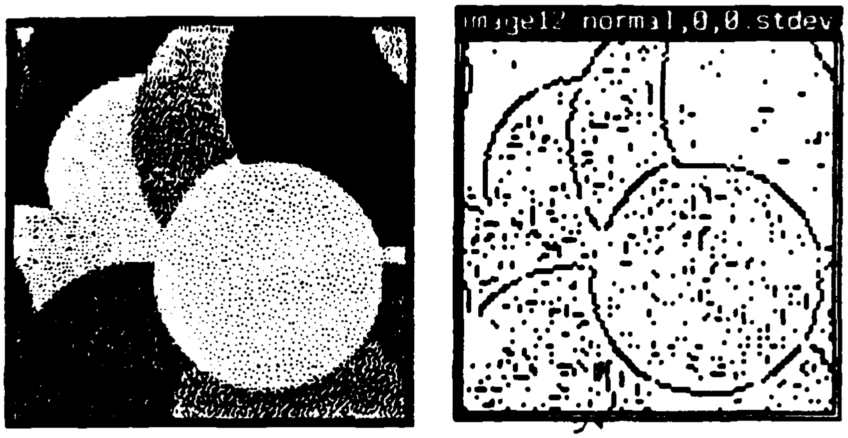


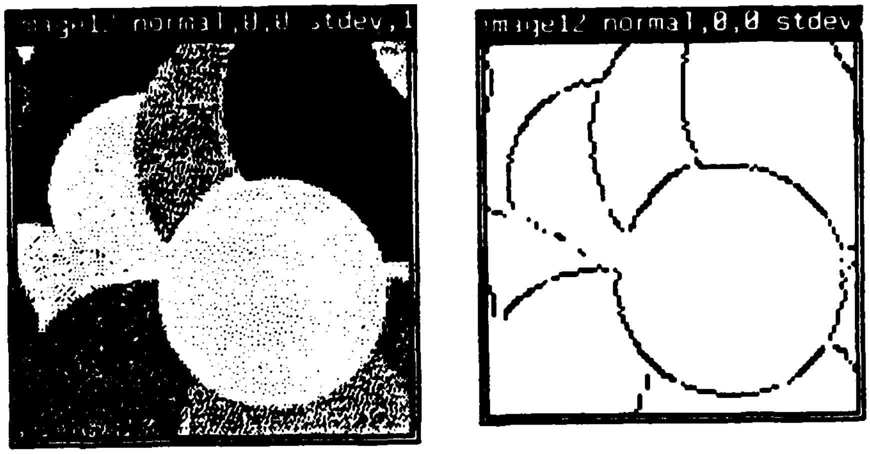
Figure 3: Artificial Test Image

The intensities of the circles were selected from a uniform distribution from 0 to 254. To the circles were added normally distributed uncorrelated noise with standard deviations 4, 8, 12, 16, 20, and 32. The software to generate images of this form was built by Myra Van Inwegen working under my direction. This software will be described in an upcoming technical report.

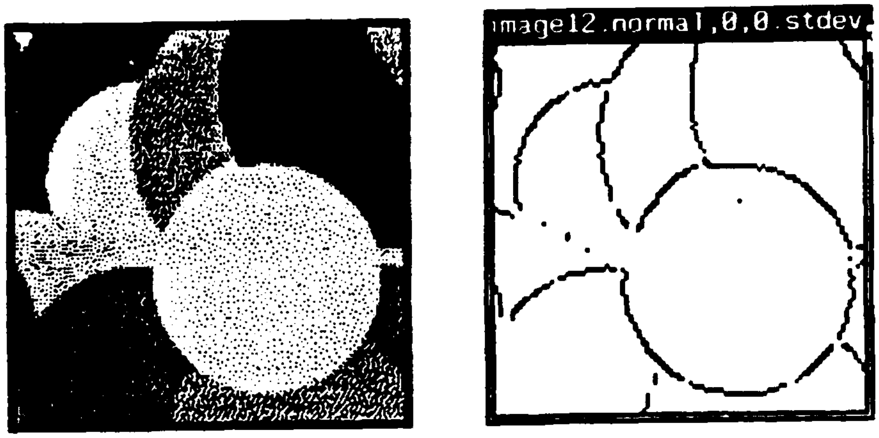
In figure 4 I show the result of applying the detector tuned to standard deviation 4 noise to the artificial image with standard deviation 12 noise added to it. In figure 5 I show the result of applying the detector tuned to standard deviation 12 noise to an image with standard deviation 12 noise added to it. In figure 6 I show the result of applying the combination of the detectors tuned to 4, 8, 12, and 16 standard deviation noise. The combination rule was that for disjoint models with the same priors. The 4 models were combined with equal probability. These operator outputs are thresholded at 0.5 probability with black indicating an edge and white indicating no edge.



a: Image with $\sigma=12$ noise b: Output of $\sigma=4$ detector
Figure 4: $\sigma=4$ detector applied to 3 image with $\sigma=12$ noise



a: Image with $\sigma=12$ noise b: Output of $\sigma=12$ detector
Figure 5: $\sigma=12$ detector applied to 3 image with $\sigma=12$ noise



a: Image with $\sigma = 12$ noise   b: Output of combined detector
Figure 6: Combined detector applied to 3 image with $\sigma = 12$ noise

Note that the result of using the combined operator is similar to that of the operator tuned to the correct noise level. Most of the false boundaries found by the $\sigma = 4$ operator are ignored by the combined operator.

Using this artificial image I have acquired statistics about the behavior of the combined detector vs the tuned ones under varying levels of noise. Figure 7 shows the false positive rate for the detector tuned to standard deviation 4 noise as the noise in the image increases[2]. Figure 8 shows the false positives for the standard deviation 12 operator. Figure 9 shows the false positive rate for the operator tuned to the current standard deviation of the noise. Figure 10 shows the false positive rate of the combined operator. Figure 11 shows the superposition of the 4 previous graphs.

[2]The operators are thresholded at 0.5 probability to make the decisions about where the boundaries are.

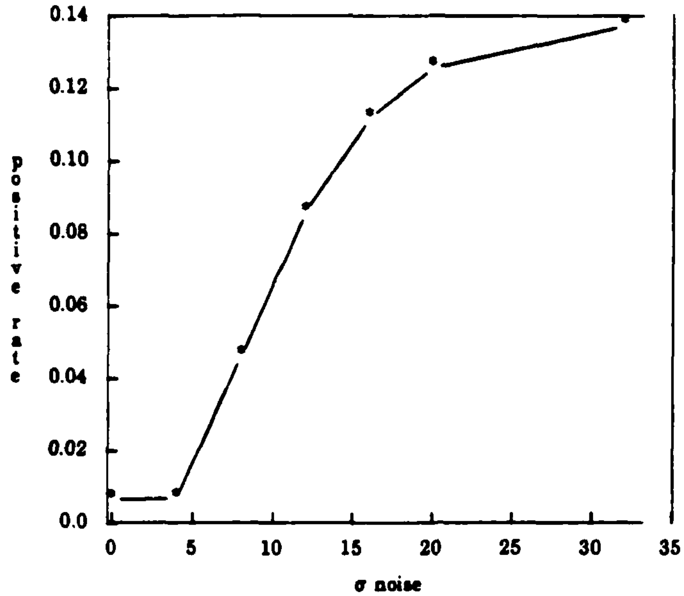


Figure 7: False positives vs noise σ for operator tuned to $\sigma = 4$ noise

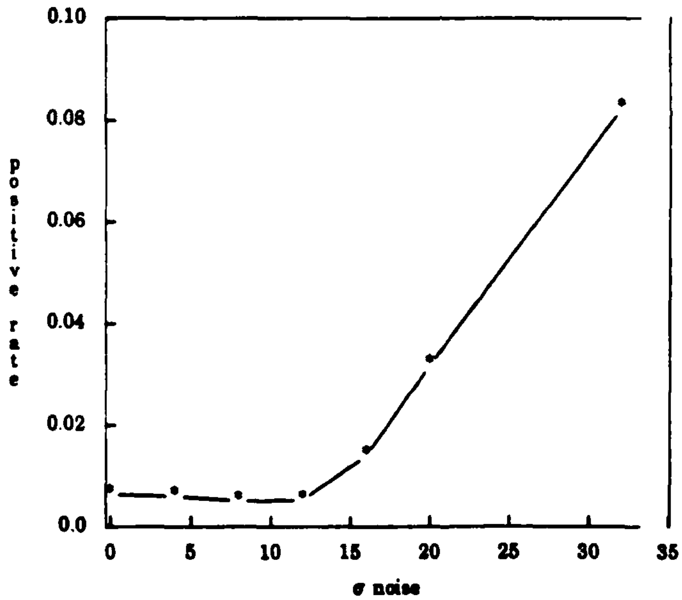


Figure 8: False positives vs noise σ for operator tuned to $\sigma = 12$ noise

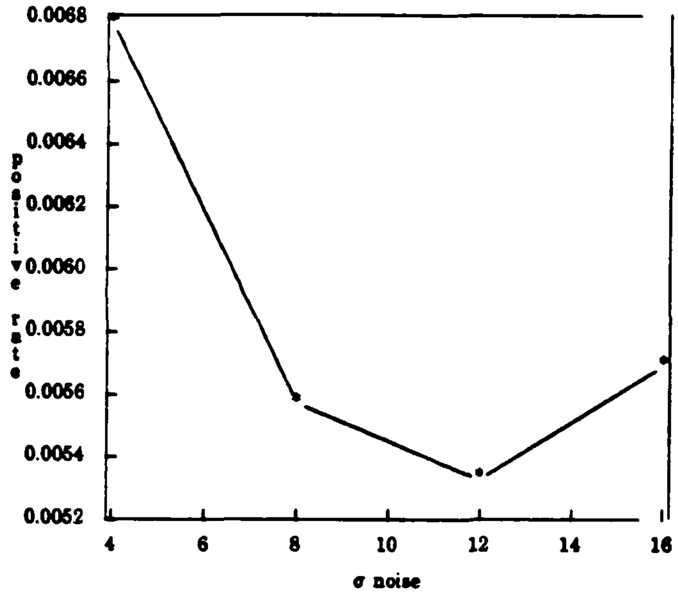


Figure 9: False positives vs noise σ for operator tuned to the noise

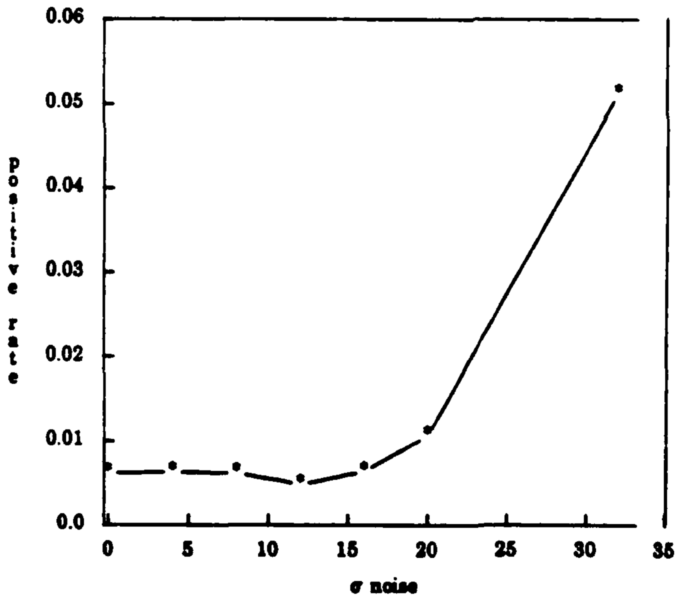
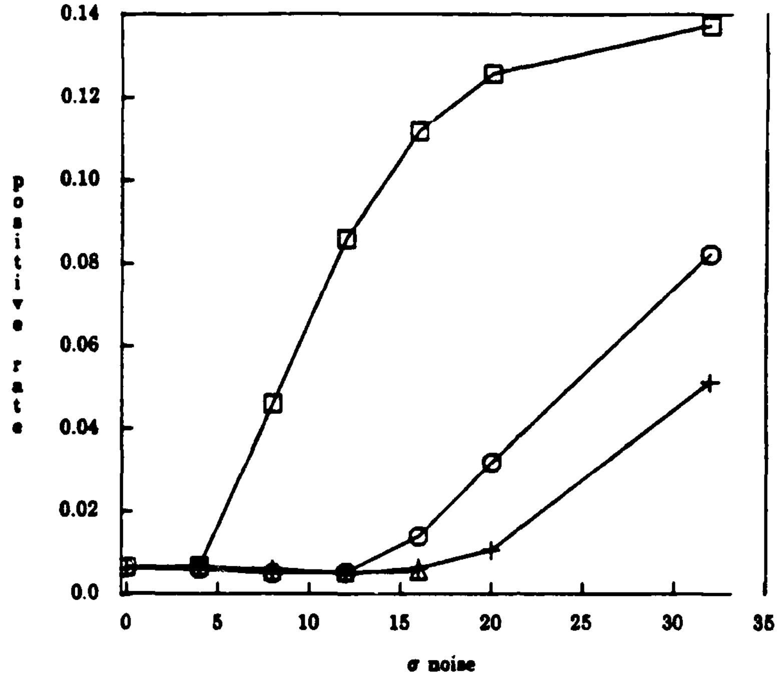


Figure 10: False positives vs noise σ for combined operator

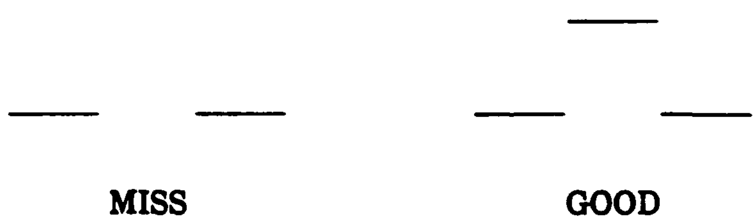


square: $\sigma = 4$ operator circle: $\sigma = 12$ operator
triangle: tuned operator cross: combined operator

Figure 11: False positives vs noise σ for all operators

Note that the combined operator has a false positive rate that is as least as good as that of the tuned operators.

I can also count false negatives. When I counted false negatives I ignored missed boundaries that had an boundary reported one pixel off normal to the boundary (because such an error is a matter of discretization rather than of a more fundamental sort). See figure 12 for an example of a 1 pixel off error.



MISS is recorded as a false negative
GOOD is recorded as a true positive

Figure 12: Example of one pixel off error

Figure 13 shows the false negative rate for the detector tuned to standard deviation 4 noise as the noise in the image increases. Figure 14 shows the false negatives for the standard deviation 12 operator. Figure 15 shows the false negative rate for the operator tuned to the current standard deviation of the noise. Figure 16 shows the false negative rate of the combined operator. Figure 17 shows the superposition of the 4 previous graphs.

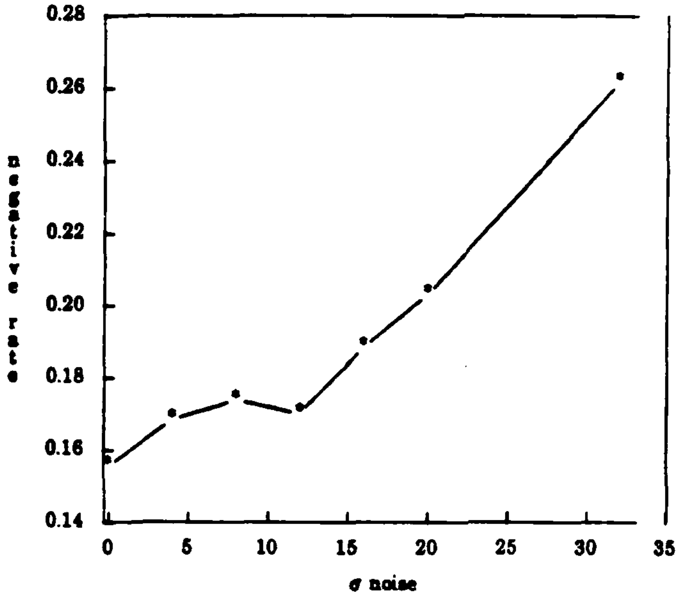


Figure 13: False negative rate for $\sigma = 4$ operator

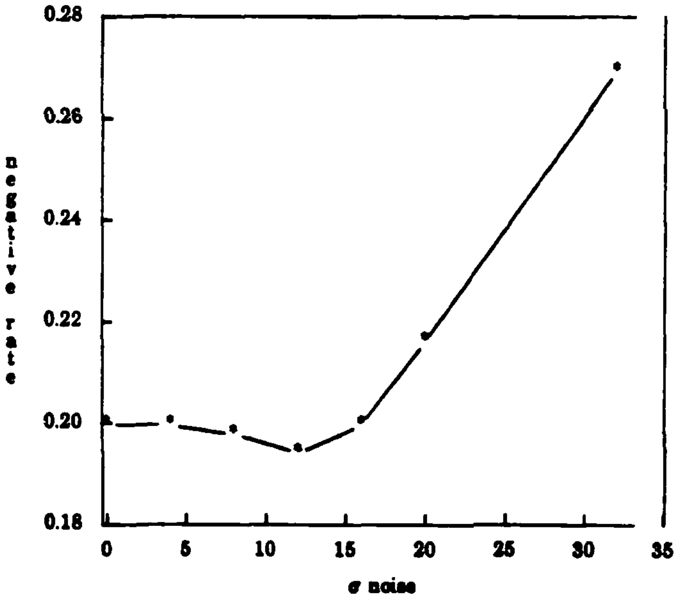


Figure 14: False negative rate for $\sigma = 12$ operator

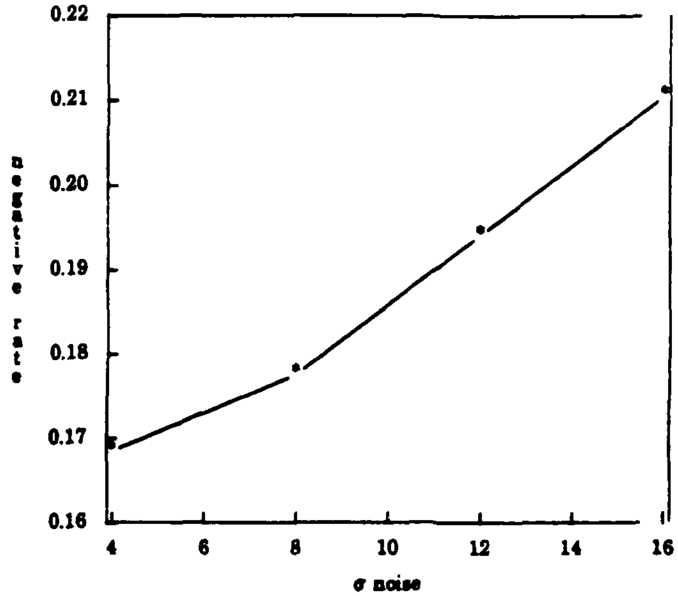


Figure 15: False negative rate for tuned operator

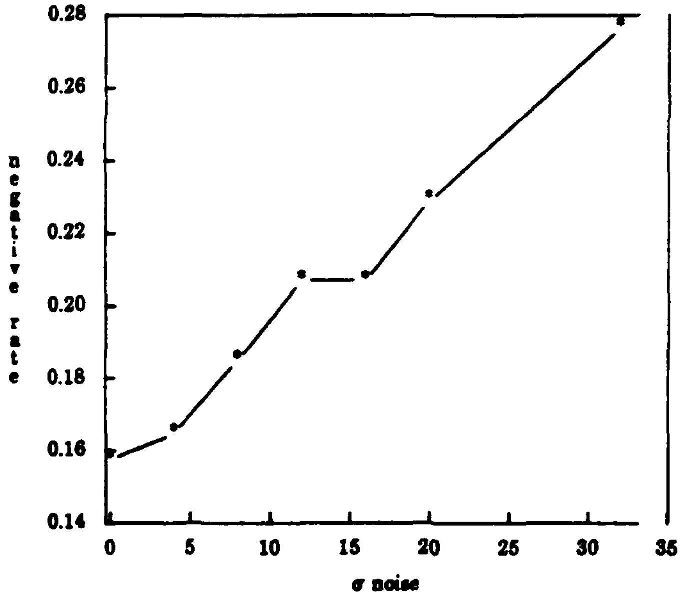
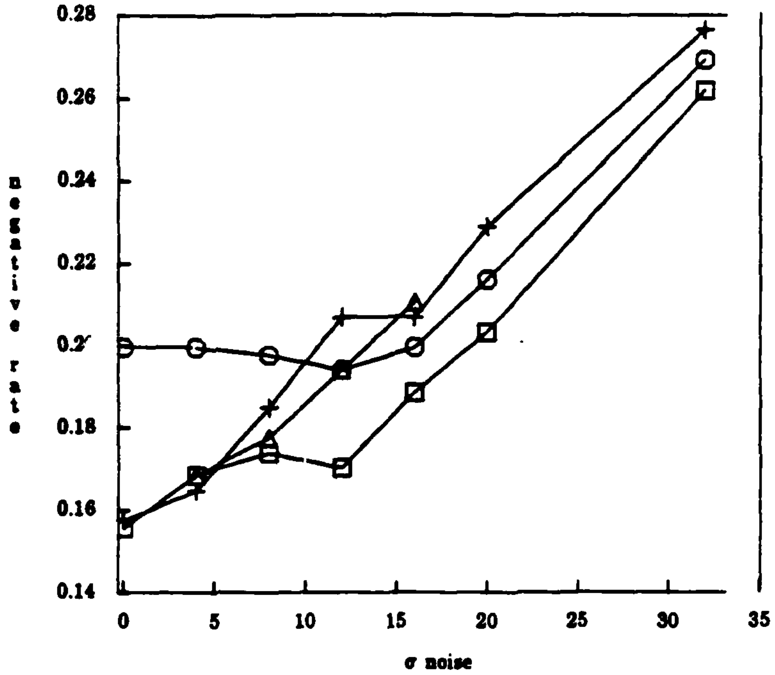


Figure 16: False negative rate for combined operator



square: $\sigma=4$ operator circle: $\sigma=12$ operator
triangle: tuned operator cross: combined operator

Figure 17: False negative rate for all operators

Here the combined operator is not always as good as the tuned operators. One must ask if this tendency of the combined operator to miss edges offsets its better performance for false positives. The next series of figures charts the total error rate for the same cases. Figure 18 shows the error rate for the detector tuned to standard deviation 4 noise as the noise in the image increases. Figure 19 shows the error rate for the standard deviation 12 operator. Figure 20 shows the error rate for the operator tuned to the current standard deviation of the noise. Figure 21 shows the error rate of the combined operator. Figure 22 shows the superposition of the 4 previous graphs.

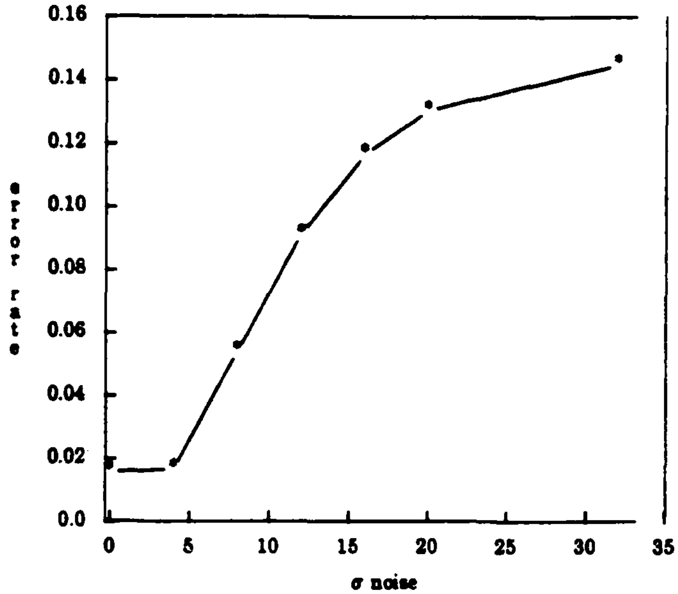


Figure 18: Total errors by the $\sigma = 4$ detector

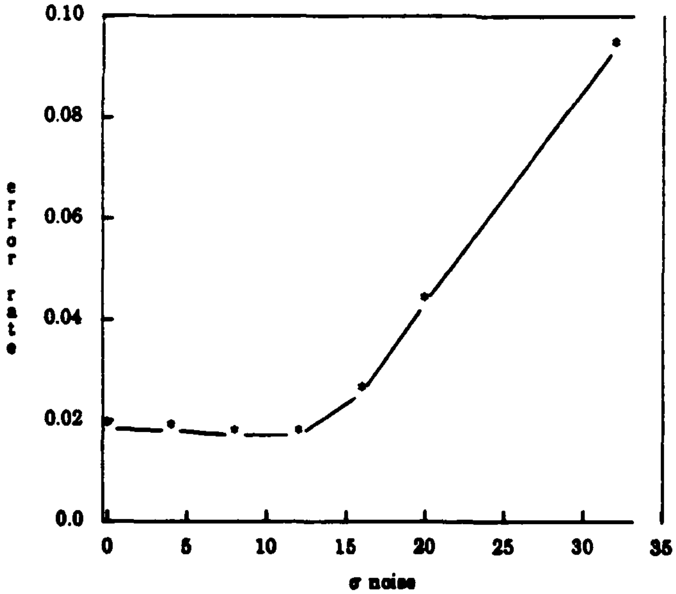


Figure 19: Total errors by the $\sigma = 12$ detector

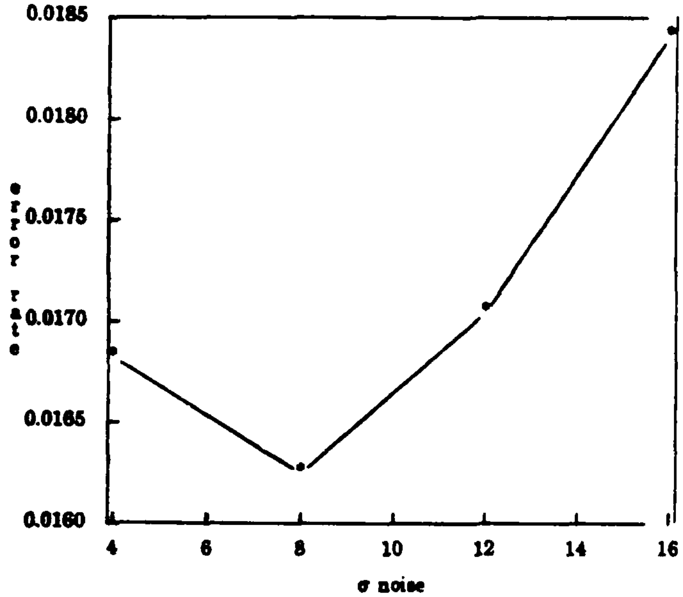


Figure 20: Total errors by the tuned detector

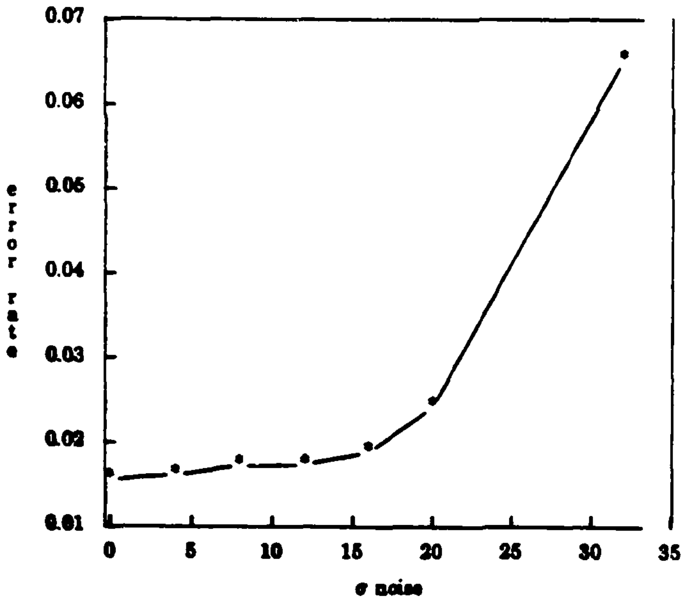
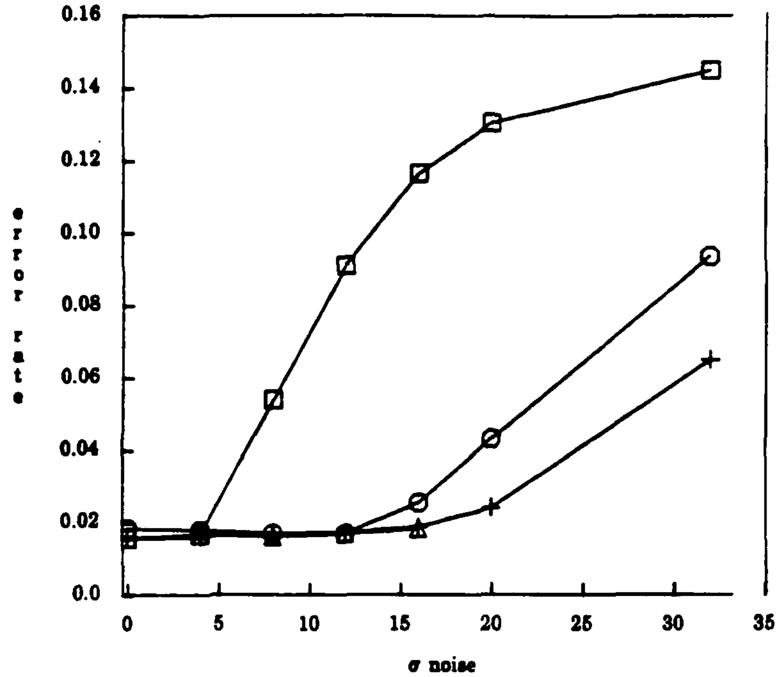


Figure 21: Total errors by the combined detector



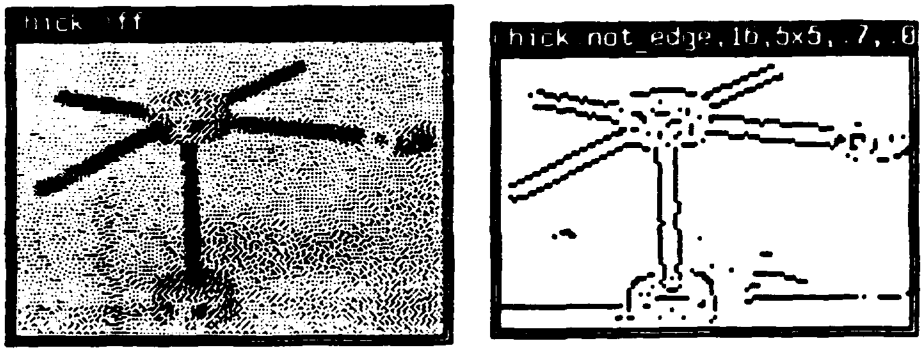
square: $\sigma=4$ operator circle: $\sigma=12$ operator
triangle: tuned operator cross: combined operator

Figure 22: Total errors by the all detectors

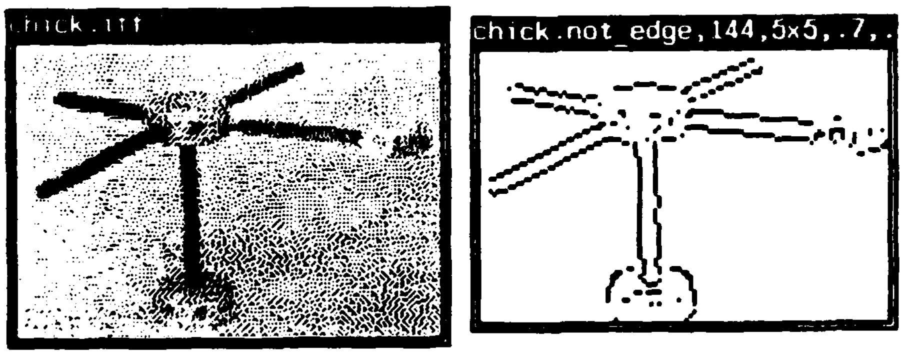
Thus the superiority of the combined operator for false positives dominates the false negative performance and the combined operator minimizes the number of errors in total. These results are evidence that my combination rule is robust.

5.2. Real Images

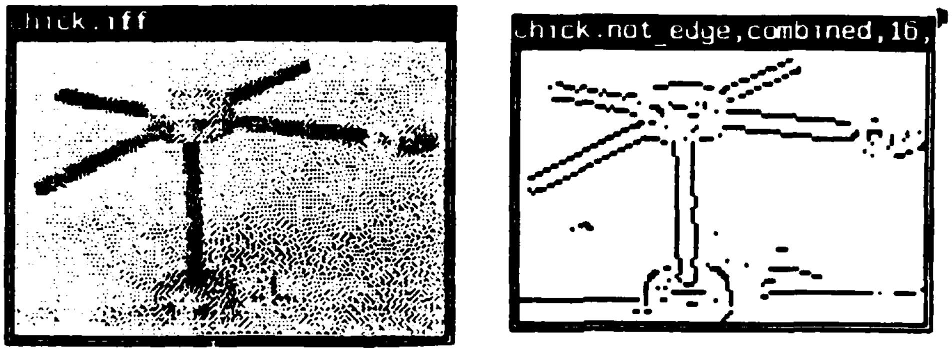
I have also tested these theories using two images taken by cameras. One of these images is a tinker toy image taken in our lab. The other is an aerial image of the vicinity of Lake Ontario. Figure 23 shows the result of the operator tuned to standard deviation 4 noise applied to the tinker toy image and thresholded at 0.5 probability. Figure 24 shows the result of the operator tuned to standard deviation 12 noise applied to the tinker toy image. Figure 25 shows the effect of combining operators tuned to standard deviation 4, 8, 12 and 16 with equal probability.



a: Tinkertoy Image b: Output of $\sigma=4$ detector
Figure 23: $\sigma=4$ detector applied to tinkertoy image



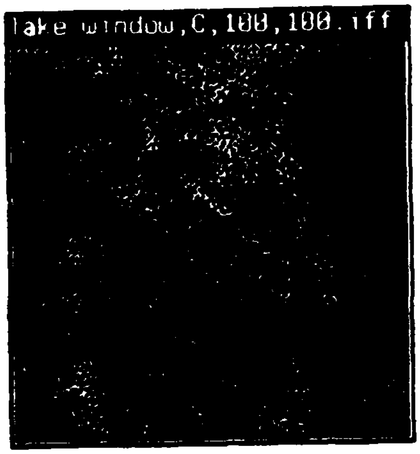
a: Tinkertoy Image b: Output of $\sigma=12$ detector
Figure 24: $\sigma=12$ detector applied to tinkertoy image

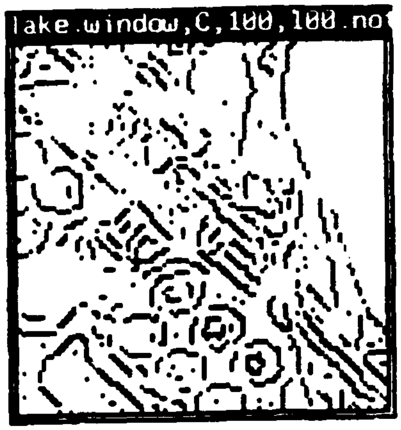


a: Tinkertoy Image b: Output of combined detector
Figure 25: Combined detector applied to tinkertoy image

Here, the result of the combined operator seems to be a cleaned up version of the standard deviation 4 operator. Most of the features that are represented in the output of the combined operator are however real features of the scene. The line running horizontally across the image that the standard deviation 4 operator and the combined operator found is the place where the table meets the curtain behind the tinkertoy. The standard deviation 4 operator was certain of its interpretation and the other operators were uncertain at that point so its interpretation was used by the combination.

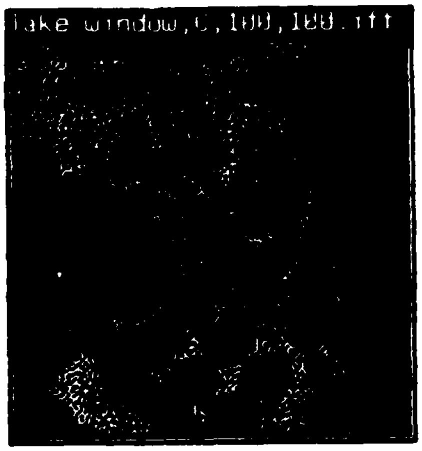
The results from the aerial image are also instructive. Figure 26 shows the result of the operator tuned to standard deviation 4 noise applied to the aerial image and thresholded at 0.5 probability. Figure 27 shows the result of the operator tuned to standard deviation 12 noise applied to the aerial image. Figure 28 shows the effect of combining operators tuned to standard deviation 4, 8, 12 and 16 with equal probability.

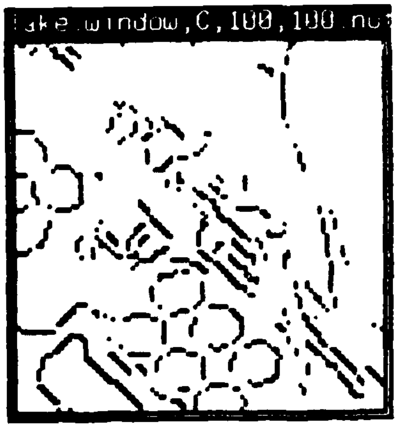




a: Aerial Image **b: Output of σ=4 detector**

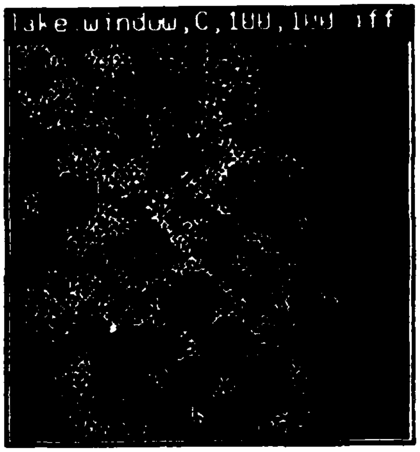
Figure 26: σ=4 detector applied to aerial image

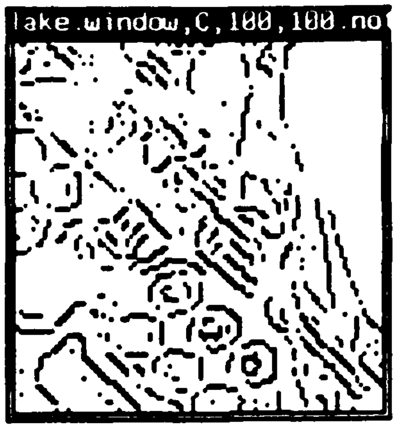




a: Aerial Image **b: Output of σ=12 detector**

Figure 27: σ=12 detector applied to aerial image





a: Aerial Image b: Output of combined detector
Figure 28: Combined detector applied to aerial image

The results from the combined operator are again a cleaned up version of the results from the standard deviation 4 operator. I believe this behavior occurs again because the features being found by the standard deviation 4 operator are in the scene. However I do not have the ground truth for the aerial image as I do for the tinkertoy image.

5.3. Future Experiments

Soon, I will apply my evidence combination rules to operators that make different assumptions about the expected image intensity histogram. The operator used so far in my experiments expects a uniform histogram between 0 and 254. Currently, a likelihood generator has been built that assumes a triangular distribution with the probability of an object having intensity less than 128 being one fourth the probability of an object having intensity greater than or equal to 128. It is not clear that the probabilities calculated based on this assumption will be significantly different from those based on the uniform histogram assumption. If there is no difference in the output of two operators the effect of combination is invisible.

Larger operators will soon be available. The likelihoods generated based on these larger operators would be finely tuned. The same evidence combination can be applied to these operators.

Likelihoods are used by Markov random field algorithms to determine posterior probabilities [Marroquin85b] [Chou87]. Likelihoods resulting from my combination rules can be used by Markov random field algorithms.

6. Previous Work

Much of the work on evidence and evidence combination in vision has been on high level vision. An important Bayesian approach (and a motivation for my work) was by Feldman and Yakimovsky [Feldman74]. In this work Feldman and Yakimovsky were studying region merging based on high level constraints. They first tried to find a probability distribution over the labels of a region using characteristics such as mean color or texture. They then tried to improve these distributions using labelings for the neighbors. Then they made merge decisions based on whether

it was sufficiently probable that two adjacent regions were the same.

Work with a similar flavor has been done by Hanson and Riseman. In [Hanson80] Bayesian theories are applied to edge relaxation. This work had serious problems with its models and the fact that the initial probabilities input were edge strengths normalized never to exceed 1. Of course such edge strengths have little relationship to probabilities (a good edge detector tries to be monotonic in its output with probability but that is about as far as it gets). In [Wesley82a] and [Wesley82b] Dempster-Shafer evidence theory is used to model and understand high level problems in vision especially region labeling. In [Wesley82b] there is some informed criticism of Bayesian approaches. In [Reynolds85] They study how one converts low level feature values into input for a Dempster-Shafer evidence system.

In [Levitt85] Tod Levitt takes an approach to managing a hierarchical hypothesis space that is baysian with some ad hoc assumptions. For the problem worked on here the paper would take weighted sums of probabilities. He does not have any way of taking an operators self confidence into account in the evidence combination. Since he was not approaching this problem in his paper I can not fault it in this respect.

There has been much use of likelihoods in recent vision work. In particular work based on Markov random fields [Geman84] [Marroquin85a] [Marroquin85b] use likelihoods. A Markov random field is a prior probability distribution for some feature of an image and the likelihoods are used to compute the marginal posterior probabilities that are used to update the field. Haralick has mentioned that his facet model [Haralick84] [Haralick86b] can be easily used to build edge detectors that return likelihoods [Haralick86a]. I also have built boundary detectors that return likelihoods and the results of using them is documented in [Sher87]. Paul Chou is using the likelihoods I produce with Markov random fields for edge relaxation [Chou87]. He is also studying the use of likelihoods for information fusion. Currently, he is concentrating on information fusion from different sources of information.

7. Conclusion

I have presented a Bayesian technique for information fusion. I show how to fuse information from detectors with different models. I presented results from applying these techniques to artificial and real images.

These techniques take several operators that are tuned to work well when the scene has certain particular properties and get an algorithm that works almost as well as the best of the operators being combined. Since most algorithms available for machine vision are erratic when their assumptions are violated this work can be used to improve the robustness of many algorithms.

References

[Aloimonos85]
J. Aloimonos and P. Chou, Detection of Surface Orientation and Motion from Texture: 1. The Case of Planes, 161, Computer Science Department, University of Rochester, January 1985.

[Ballard82]
D. H. Ballard and C. M. Brown, in *Computer Vision*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1982, 125.

[Chou87] P. Chou, Multi-Modal Segmentation using Markov Random Fields, *Submitted to IJCAI*, January 1987.

[Feldman74]
J. A. Feldman and Y. Yakimovsky, Decision Theory and Artificial Intelligence: I. A Semantics-Based region Analyzer, *Artificial Intelligence* 5(1974), 349-371, North-Holland Publishing Company.

[Frieden85]
B. R. Frieden, Estimating Occurrence Laws with Maximum Probability, and the Transition to Entropic Estimators, in *Maximum-Entropy and Bayesian Methods in Inverse Problems*, C. R. Smith and W. T. G. Jr. (editor), D. Reidel Publishing Company, Lancaster, 1985.

[Geman84]
S. Ge n and D. Geman, Stochastic Relaxation, Gibbs Distributions, and the Bayesian Resto ion of Images, *PAMI 6*,6 (Novermber 1984), 721-741, IEEE.

[Good50] I. J. Good, *Probability and the Weighing of Evidence*, Hafner Publishing Company., London, New York, 1950

[Good83] I. J. Good, Subjective Probability as the Measure of a Non-measurable Set, in *Good Thinking: The Foundations of Probability and its Applications*, Minneapolis (editor), University of Minnesota Press, Minneapolis, 1983, 73-82.

[Hanson80]
A. R. Hanson, E. M. Riseman and F. C. Glazer, Edge Relaxation and Boundary Continuity, 80-11, University of Massachusetts at Amherst, Computer and Information Science, May 1980.

[Haralick84]
R. M. Haralick, Digital Step Edges from Zero Crossing of Second Directional Derivatives, *PAMI 6*,1 (January 1984), 58-68, IEEE.

[Haralick86a]
R. Haralick, Personal Communication, June 1986.

[Haralick86b]
R. M. Haralick, The Facet Approach to Gradient Edge Detection, *Tutorial 1 Facet Model Image Processing (CVPR)*, May 1986.

[Horn70] B. K. P. Horn, *Shape from Shading: A Method for FInding the SHape of a Smooth Opaque Object from One View*, Massachusetss Institute of Technology Department of ELectrical Engineering., August 1970

[Ikeuchi80]
K. Ikeuchi, Shape form Regular Patterns (an Example of Constraint Propagation in Vision), 567, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, March 1980.

[Johnson85]
R. W. Johnson and J. E. Shore, Introduction to Minimum-Cross-Entropy Spectral Analysis of Multiple Signals, in *Maximum-Entropy and Bayesian Methods in Inverse Problems*, C. R. Smith and W. T. G. Jr. (editor), D. Reidel Publishing Company, Lancaster, 1985.

[Levitt85] T. S. Levitt, Probabilistic Conflict Resolution in Hierarchical Hypothesis Spaces, *Proceedings: Uncertainty and Probability in Artificial Intelligence*, August 14-16, 1985, 265 272.

[Marroquin85a]
J. Marroquin, S. Mitter and T. Poggio, Probabilistic Solution of Ill-Posed Problems in Computational Vision, *Proceedings: Image Understanding* Workshop, December 1985, 293-309. Sponsored by: Information Processing Techniques Office Defence Advanced Research Projects Agency.

[Marroquin85b]
J. L. Marroquin, Probabilistic Solution of Inverse Problems, Tech. Rep. 860, MIT Artificial Intelligence Laboratory, September 1985.

[Ohlander79]
R. Ohlander, K. Price and D. R. Reddy, Picture Segmentation using a Recursive Region Splitting Method, *CGIP 8*,3 (1979).

[Reynolds85]
G. Reynolds, D. Strahman and N. Lehrer, Converting Feature Values to Evidence, *PROCEEDINGS: IMAGE UNDERSTANDING WORKSHOP*, December 1985, 331-339. Sponsored by: Information Processing Techniques Office, Defence Advanced Research Projects Agency.

[Sher86] D. Sher, Optimal Likelihood Detectors for Boundary Detection Under Gaussian Additive Noise, *IEEE Conference on Computer Vision and Pattern Recognition*, Miami, Florida, June 1986.

[Sher87] D. B. Sher, Advanced Likelihood Generators for Boundary Detection, TR197, University of Rochester Computer Science Department, London, England, January 1987. Submitted in shorter form to International Conference on Computer Vision.

[Wesley82a]

L. P. Wesley and A. R. Hanson, The Use of an Evidential-Based Model for Representing Knowledge and Reasoning about Images in the Visions System, *PAMI 4*,5 (Sept 1982), 14-25, IEEE.

[Wesley82b]

L. P. Wesley and A. R. Hanson, The use of an Evidential-Based Model for Representing Knowledge and Reasoning about Images in the VISIONS System, *Proceedings of the Workshop on Computer VIsion: Representation and Control*, August 1982, 14-25.

Appendix B-8

Optimal Likelihood Generators for Edge Detection under Gaussian Additive Noise

David Sher
Computer Science Department
The University of Rochester
Rochester, New York 14627

TR 185

August 1986

A technique is presented for determining the probability of an edge at a point in an image. The image is modeled as an ideal image that is convolved with a linear blurring function and also with uncorrelated Gaussian additive noise. The ideal image is modeled by a set of templates for local neighborhoods. Every neighborhood in the ideal image is assumed to fit one of the templates with high probability. A computationally feasible scheme to compute the probability of edges is given. The output of several of the likelihood generators based on this model can be combined to form a more robust likelihood generator using the results described in *Developing and Analyzing Boundary Detection Operators Using Probabilistic Models* presented in the first Workshop on Probability and Uncertainty in Artificial Intelligence by the author [13].

This work would have been impossible without the advice and encouragement of Chris Brown my thesis advisor. This work was supported in part by the Defense Advanced Research Projects Agency U. S. Army Engineering Topographics Lab. grant number DACA76-85-C-0001 and The National Science Foundation. grant number DCR-8320136.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR 185	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Optimal Likeilhood Generators for Edge Detection under Gaussian Additive Noise		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) David Sher		8. CONTRACT OR GRANT NUMBER(s) DACA76-85-C-0001
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Department The Unviersity of Rochester Rochester, New York 14627		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS DARPA/1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE
		13. NUMBER OF PAGES 10
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Office of Naval Research Information Systems Arlington, VA 22217		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

edge detection, template, likelihood, Bayesian reasoning

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A technique is presented for determining the probability of an edge at a point in an image that is convolve with a linear blurring function and also with uncorrelated Gaussian additive noise. The ideal image is modeled by a set of templates for local neighborhoods. Every neighborhood in the ideal image is assumed to fit one of the templates with hight probability. A computaionally feasible scheme to compute the probability of edges is given. The output of several of the likelihood generators based on this model can be

20. ABSTRACT (Continued)

Combined to form a more robust likelihood generator using the results decribed in Developing and Analyzing Boundary Detection Operators Using Probabilistic Models presented in the first Workshop in Probability and Uncertainty in Artificial Intelligence by the author(13).

1. Edge Detection: The Problem and Previous Approaches

The major problem of low-level vision is that images are ambiguous: two different scenes can result in the same image. The major source of ambiguity that I am concerned with is noise. Noise is generally the result of imperfections of the sensors used to produce the image. Because of noise the same scene can result in any observed image whatsoever. It is much more likely however to result in some images than others. My work is about techniques for combating noise and the resulting ambiguity and thus is applicable to vision tasks where noise presents a significant problem.

My approach to low level vision is unusual for such research. Consider the problem of segmentation, in particular, consider the problem of finding regions of uniform reflectance. The image is modeled as a set of regions of constant reflectance with occlusion boundaries between them. Most approaches to this problem try to return an answer that is best, in the sense that the probability of the given answer differing from the correct answer in a significant way is minimized. Such an algorithm applies *estimation theory* to the problem of low level vision.

Instead, this paper derives algorithms that attempt to calculate the probability of a boundary passing between two points. In low-level vision usually one can acquire a sufficiently specific model for the probability to be uniquely defined, even through the image is ambiguous. One advantage of this approach is that a variety of different estimates of the segmentation can be derived from these probabilities by simple operations.

This paper concentrates on the problem of deriving the probability of a boundary from a window on the image. Classically this task has been called edge detection. I am using a template based model for this work: It is assumed that if the image was viewed through a noiseless sensor then every window on the image would match one element of a set of templates. Since the image wasn't produced by noiseless sensors its windows look like some template followed by noise according to the model.

Recently two works have been published that take an approach similar to mine. One that is similar is by Art Owen [12] on pixel classification for Landsat images. The operator he derives returns likelihoods for neighborhoods instead of pixels. Owen's work uses a somewhat more sophisticated model to derive his priors (a Poisson model of boundaries). The work has no noise model and does not consider combination rules. Likelihoods are derived by training on test cases. Owen can use training to get his likelihoods because of the small number of categories he uses and because he uses binary (thresholded) images. This reduces the number of cases he had to deal with so the operator can be conveniently trained.

Another work that takes an approach similar to mine is that of Li and Dubes [9] on matching small templates in binary images. They use Neyman-Pierson statistics. Neyman-Pierson statistics are used because there is a well defined null hypothesis (the object is not in the scene). Li and Dubes derive a likelihood ratio test. Such a test has maximal power if it is based on a complete and sufficient statistic. The way they derive the likelihood ratio is to derive likelihood generators. They approximate the likelihoods deriving operators much in the same spirit that I derive mine in section 3.

There has been some work on using Bayesian techniques (techniques using likelihoods and prior probabilities) to estimate edge positions. In particular the work described in [3] and [6] use Bayesian techniques for image reconstruction and [8] uses Bayesian technique for reconstruction and edge detection (as a side effect). These techniques have the weakness that they look for the maximum *a posteriori* likelihood (the *MAP* assumption). The MAP assumption only holds when a small set of answers are the only ones acceptable as correct with 0 loss and all other segmentations have the same loss (1 loss). I believe that a 0-1 loss function is unrealistic for most applications. A 0-1 loss function is realistic if getting a boundary wrong at a single point is as bad as getting it wrong everywhere, because both possibilities result

in 1 loss according to the 0-1 loss function. In low-level vision the usefulness of an estimate drops off gradually as errors accumulate. Some good results have been gained using these techniques.

Much work has been done using signal detection theory for deriving operators. However most work based on signal detection theory is limited to operators that compute linear functions on the image. Because of this limitation the operators generated are the optimal linear operators given a figure of merit. In particular the Wiener filter is optimal for reconstructing images given a least squares cost function and a correct noise model and image model. [1].

Canny [5] has developed an operator that is optimal according to a figure of merit that contains detection and localization. He limited himself to linear shift invariant operators. His operators looked a great deal like difference of gaussian operators. He modeled edges as a template and developed a technique for generating an operator for an arbitrary template. I intend to generate optimal detectors under my system with the same models.

Canny [5] Lunscher and Beddoes [10] and Torre and Poggio [15] limit the class of functions that they consider for edge detection to linear shift independent operators. Thus their operators are convolutions. When they indicate that their operators are optimal they mean that they do the best job for functions in the class of linear shift independent operators. The class of functions I use is the class of functions of a window on the image. Such operators are shift independent but they are not necessarily linear. The optimal operator from this class theoretically is the best possible edge detector for a specified window size.

Much of the work done in computer vision has been developed with idiosyncratic objectives. Because of the their objectives differed from mine the algorithms some people developed have serious shortcomings from my viewpoint. One alternate set of objectives is those held by researchers inspired by biological modeling. An excellent work in biological modeling is that of Fleet [7]. His work is on the temporal and spatial characteristics of center-surround operators. Torre and Poggio's work [15] also is of this form.

When working on modeling one tries to develop algorithms whose behavior closely approximates that of a human vision system. An example of such approximation is to have only band limited operators because the cells on the mammalian optic nerve have been shown to be band limited. I only band limit operators if it is shown that the phenomena being detected are band limited or that a band limited operator is sufficient to detect the phenomena without loss of accuracy.

Much work has been done on segmentation without considering optimality or probability. A summary of work on edge detection and relaxation occurs in [4] Recently some good work on edge detection has been done by Canny [5] and Nalwa [11].

2. The Image Model

In the image restoration literature much work has been done on a particular form of noise. The noise introduced by the sensor is modeled by a linear blurring function followed by gaussian additive mean 0 noise [2]. The log image from a photograph has gaussian additive noise in its linear region from the randomness inherent in film grain. Gaussian additive noise occurs in any system whose noise is a result of many small perturbations added together (by the central limit theorem). Blur can result from vibrations in the camera, motion in the scene and the physics of light. I make a standard simplification in that I assume the blur is linear and shift invariant. Blur from vibrations in the camera and the physics of light has this property. Blur from motion in the scene tends to be linear and shift invariant within a rigid object. Thus I

model the noise as convolving the image with a blur function and then adding a gaussian additive mean 0 random factor.

I also need a model of an image to derive a likelihood generator. A *likelihood generator* is an intermediate stage in an algorithm that calculates the probability of a boundary at a point. More details on likelihood generators are in the next section.

Here, I derive the optimal likelihood generator that looks at a window in the image. Thus I need only model windows in the ideal image. I model the ideal image as consisting of windows that each match an element of a set in a set of sets of templates. Thus if I can derive the likelihood of the observed window given that its ideal counterpart matches each template in a set and the a priori probability of each template then I can derive the likelihood of the window belonging to the set of templates. As an example consider the set of templates that consist of a uniform intensity (figure 1).

Figure 1: A template of uniform intensity.

100	100	100	100

This set of templates models the interior of a region of uniform intensity. Consider what an occlusion edge between two such regions looks like. Such an event can be modeled by a template of the form in figure 2.

Figure 2: A template of a step edge.

100	100	200	200

This template is often called a step edge in the edge detection literature. I also need to model the event that there is an off center edge in the window. I call this event a *near edge* event. The near edge events are modeled by templates like those of figure 3.

Figure 3: Templates for a near edge.

100	200	200	200

100	100	100	200

So 3 useful sets of templates are templates like those in figure 1, 2, and 3 with all possible intensities substituted for 100 and 200. These templates model all possible configurations of a 1 by 4 window in an ideal image where all regions are at least 3 pixels wide. If I can derive the likelihood of an observed window having a counterpart in each of these sets then I can derive the probability of a boundary in the middle of the window using Bayes' law (see next section).

3. Likelihood Generators

Often it is easier to state and solve the inverse vision problem (which is why computer graphics can generate realistic images that current image understanding systems can't analyze). For low level vision it is easier to describe the probable structure of an observed intensity image in the presence of a boundary than to describe the probability distribution on the boundary given an observed image. In particular the models described in the previous section have this property.

The probability that the observed window's pixels are assigned a set of values a when a feature f takes on value v is the *likelihood* of v for a. I use $L_f(a \mid v)$ as shorthand notation for the likelihood. A *likelihood generator* is an algorithm that uses a model D to estimate the likelihood of v for a. Thus I use $L_f(a \mid v \& D)$ as notation for the output of a likelihood generator. Given a likelihood generator for D and a prior estimate of the distribution of f's values then one can make a feature detector for f using Bayes'

Rule:

$$P_f(v \mid a\&D) \equiv \frac{L_f(a \mid v\&D)prior_f(v)}{\sum_{v' \in V} L_f(a \mid v'\&D)prior_f(v')} \tag{1}$$

I call the feature detector thus derived a *Bayesian feature detector* for model D.

The set of likelihoods for a feature f given an observation a contains more information than (1) uses. The denominator in (1)

$$\sum_{v' \in V} L_f(a \mid v'\&D)prior_f(v') \tag{2}$$

is the probability that a would occur given the prior estimate of the distribution on f's feature space. If the probability is too low then the model being used probably is not correct. I use this information combined with *a priori* information about the reliability of the model to derive an evidence theory in [14].

4. Likelihoods for a Single Template

The problem I address in this paper is to find the likelihood of an observed window given a template and a model for the noise. Let $O=\left\{o_i\right\}$ represent the window that was observed. Let $T=\left\{t_j\right\}$ represent the template. Then I need $P(O \mid T\&D(\sigma,B))$ where σ is the standard deviation of the gaussian mean 0 additive noise and B represents the blurring function. Assume that B is negligible outside a window of size (w_B,l_B) pixels and the template is of size (w_T,l_T). Then the effect of the blurring function $B \otimes T$ ($\otimes$ is correlation where the template never falls beyond the window's edge, $X \otimes X$ is a single number that is the sum of squares of X's elements) is completely determined in a region of size (w_T-w_B+1, l_T-l_B+1) pixels (see figure 4).

Figure 4: Effect of a Blur Function on a Template.

T:

100	100	100	200	200	200

$\otimes$

B:

.25	.5	.25

=

$T \otimes B$:

100	125	175	200

I assume in the rest of this paper that the observation window O lies completely within the determined region. So the only remaining probabilistic element is the gaussian additive noise. If I is the identity function then I need to determine $P(O \mid T \otimes B\&D(\sigma,I))$. I refer to the elements of $T \otimes B$ as the set $\left\{t'_k\right\}$.

Since the only noise left in the problem is the uncorrelated gaussian additive noise (since blur has been handled) the likelihood is the product of the likelihoods at each pixel.

$$P(O \mid T \otimes B\&D(\sigma,I)) = \prod_i P(o_i \mid t'_i\&D(\sigma,I)) \tag{3}$$

Since the noise is gaussian the likelihood at a point has this form:

$$P(o_i \mid t'_i \& D(\sigma, I)) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[\left[-\left[o_i - t'_i\right]^2\right] / \left[2\sigma^2\right]\right] \tag{4}$$

Thus the equation for the likelihood of the window can be stated as:

$$P(O \mid T \otimes B \& D(\sigma, I)) = \frac{1}{\left[\sqrt{2\pi}\sigma\right]^n} \exp\left[\left[-\sum_i \left[o_i - t'_i\right]^2\right] / \left[2\sigma^2\right]\right] \tag{5}$$

The likelihood can be restated mathematically as:

$$\frac{1}{\left[\sqrt{2\pi}\sigma\right]^n} \frac{\exp\left[\left[O \otimes (T \otimes B)\right] / \left[\sigma^2\right]\right]}{\mathrm{En}(O, \sigma)\mathrm{En}(T \otimes B, \sigma)} \tag{6}$$

Where $\mathrm{En}(X, \sigma)$ is $e^{(X \otimes X)/(2\sigma^2)}$ which I refer to as the energy of X relative to σ. Note that $\mathrm{En}(O, \sigma)$ is independent of the template while $\mathrm{En}(T \otimes B, \sigma)$ is independent of the observed window. These results mean that $\mathrm{En}(T \otimes B, \sigma)$ can be precomputed while the cost of computing $\mathrm{En}(O, \sigma)$ can be amortized over the entire set of templates.

5. Likelihoods for Sets of Templates

Here, I examine efficiently calculating the likelihood of a set of templates given an observed image. In particular I examine the set of templates whose elements are all linear functions of a characteristic template, T_0. Thus I describe such a set as $aT_0 + b$. I call such a set a *linear set* of templates. The set of step edges with a fixed step point can be described as a linear set. The set of symmetric peak edges are linear functions of a prototypical peak edges hence are a linear set. The linear slopes are linear functions of the function $f(x) = x$ hence are a linear set too.

I limit my blur functions to blurs that leave uniform intensity images unchanged. Then my set of $B \otimes T$ is of the form $aB \otimes T_0 + b$. The likelihood of the observed image given a member of a linear set is:

$$\frac{1}{\left[\sqrt{2\pi}\sigma\right]^n} \frac{\exp\left[\left[2(aO \otimes (T_0 \otimes B) + bO \otimes I)\right] / \left[2\sigma^2\right]\right]}{\left[\mathrm{En}(O, \sigma)\right]\left[\mathrm{En}(aT_0 \otimes B + b, \sigma)\right]} \tag{7}$$

The triplet $(\mathrm{En}(O, \sigma), O \otimes T_0 \otimes B, O \otimes I)$ is sufficient for determining the likelihood of this set of templates. The class of templates is indexed by a and b. To find the likelihood I need *a priori* probabilities for the different templates. I describe these probabilities with $P_{T_0}(a, b)$.

The likelihood of a linear set is:

$$\frac{1}{\left[\sqrt{2\pi}\sigma\right]^n \mathrm{En}(O, \sigma)} \sum_{a,b} \frac{\exp\left[\left[(aO \otimes (T_0 \otimes B) + bO \otimes I)\right] / \left[\sigma^2\right]\right]}{\mathrm{En}(aT_0 \otimes B + b, \sigma)} P_{T_0}(a, b) \tag{8}$$

Let F_{T_0} be defined in equation (9).

$$F_{T_0}(C,S) = \sum_{a,b} \frac{\exp\left\{\left[(aC+bS)\right] / \left[\sigma^2\right]\right\}}{\mathrm{En}(aT_0 \otimes B + b, \sigma)} P_{T_0}(a,b) \tag{9}$$

Then equation (7) can be rewritten as equation (10).

$$\frac{1}{\left[\sqrt{2\pi}\sigma\right]^n \mathrm{En}(O,\sigma)} F_{T_0}(O \otimes (T_0 \otimes B), O \otimes I) \tag{10}$$

This implies an algorithm for deriving the likelihood of a linear set of templates.

Let V be the variance of the noise σ^2
Let K be $\left[\sqrt{2\pi}\sigma\right]^n$
For each window W in the image do

{
- S: Let S be the sum of the pixels in W
- SS: Let SS be the sum of the squared pixels in W
- C: Let C be the correlation of W with the $T_0 \otimes B$
- F: Let F be $F_{T_0}(C,S)$
- E: Let E be exp(SS / (2*V))
- O: Output F / (K * E)

}

If there are N pixels in the image steps S and SS require $O(N)$ operations counting adds and multiplies. Step C requires $O(N\log N)$ operations. Steps E and O are also $O(N)$ operations steps. Thus the algorithm requires $O(N\log N)$ operations plus whatever is required to execute step F. I propose to calculate F_{T_0} by table-lookup on the values of S and C. Thus step F is just a table-lookup.

The size of the table that holds F is the product of the number of possible values of C and S. Both of these can be calculated given the number of gray-levels in the image, G, and the number of pixels in the window, n, and $T_0 \otimes B$. The number of possible values for S is nG and the number of values that C can be is $G(T_0 \otimes B)$. Thus the number of elements in the table is $n(T_0 \otimes B)G^2$.

For a central step edge with a 1 by 8 window $n=8$ and $T_0 \otimes B=4$. Thus the size of the table is $32G^2$. Table 1 is a table of G values and resulting table sizes.

Table 1: Table Sizes to calculate F_{T_0}

G	Number of Table Entries	Storage for Table in bytes (in double precision)
4	512	4K
16	8192	64K
64	131072	1M
256	2097152	16M

The more gray-levels the more difficult it becomes to store the table. It also becomes more work to calculate the entire table. Thus to handle 64 or more gray-levels I suggest that a smaller table be used with interpolation. If there are symmetries in F_{T_0} a smaller table is sufficient to store the function. As an example if $F_{T_0}(S,C) = F_{T_0}(S+16,C)$ then only F_{T_0} need only be calculated for S between 1 and 16. At this moment no such symmetries have been discovered.

6. Detecting 1-D Step Edges Optimally

For the model of regions of uniform intensity with step edges between them I need only calculate the likelihoods for two linear sets of templates. One template is the uniform intensity template. The likelihood of this template can be calculated from the standard deviation of the observed window. The other is the step edge template with the step in the middle. If I have a prior on the probability of a boundary then I have the tools necessary to build an optimal edge detector for my model.

The near edge templates can be approximated by the likelihoods calculated at the neighboring (overlapping) windows for the central step edge linear set. Since I am deriving a 1 dimensional edge detector, the likelihood of an edge in the center of an overlapping window is the likelihood of an edge directly to the right or the left of the center of the window. In the step edge model all regions are at least $w/2$ pixels wide given a template width of w. Thus the near edge events are exclusive of the central edge events.

I assume a cost function that simply counts the number of points mislabeled as boundaries or nonboundaries when the opposite is the case. The prior probability of a central edge and any near edge event is equal under models that do not have a prefered position for objects. Thus if the likelihood of a central edge is not maximal among all the overlapping windows then the optimal estimate does not have an edge at this point. Only local maxima among the likelihood of step edge function are reported. Thus multiple reporting of an edge is precluded. Also only edges that satisfy the inequality (11) are reported:

$$P(O \mid E)P_E > P(O \mid U)P_U \tag{11}$$

where E represents the event that there is an edge in the center of the window and P_E is the prior probability of that event while U represents the event that there is no edge anywhere in the window and P_U is the prior probability of that event.

I can also use my work on evidence combination to combine likelihood generators that make different assumptions about the noise and blur. Many of the operations I use to evaluate the likelihood of a linear set of templates under one kind of noise can be used for many different kinds of noise. As an example $En(O,\sigma)$ is used by all likelihood generators based on linear sets of templates. Also all templates that have the same value for $En(T_0 \otimes B, \sigma)$ and $(T_0 \otimes B) \otimes I$ and have the same values for P_{T_0} can share the same table to calculate F_{T_0} since it depends only on these parameters. Thus if all the differently oriented edge templates have the same sum of pixel values and the same sum of squares of pixel values they can share the same table for F_{T_0}.

7. Conclusions

In this paper I demonstrated an algorithm for edge detection that is mathematically optimal for a popular model. Since F_{T_0} is increasing in $O \otimes (T_0 \otimes B)$ this algorithm thresholds using a function of the sum of the pixels in the window and the sum of the squares of the pixels in the window. The algorithm only reports an edge if there are no nearby edges with greater likelihood. That test is similar to edge thinning in standard work. Thus the algorithm is similar to algorithms that run a thresholded convolution and then thin. Currently this algorithm is being implemented and experimental results will soon be forthcoming.

References

[1] H. C. Andrews and B. R. Hunt, *Digital Image Restoration*, 126-147 , Prentice-Hall, INC., Englewood Cliffs, New Jersey 07632, 1977

.

[2] H. C. Andrews and B. R. Hunt, *Digital Image Restoration*, 8-26 , Prentice-Hall, INC., Englewood Cliffs, New Jersey 07632, 1977

.

[3] H. C. Andrews and B. R. Hunt, *Digital Image Restoration*, 187-211 , Prentice-Hall, INC., Englewood Cliffs, New Jersey 07632, 1977

.

[4] D. H. Ballard and C. M. Brown, in *Computer Vision*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1982, 14,63,85-86.

[5] J. F. Canny, Finding Edges and Lines in Images, 720, MIT Artificial Intelligence Laboratory, June 1983.

[6] H. Derin, H. Elliott, R. Cristi and D. Geman, Bayses Smoothing Algorithms for Segmentation of Binary Images Modeled by Markov Random Fields, *PAMI* 6.6 (Novermber 1984), 707-720, IEEE.

[7] D. J. Fleet, The Early Processing of Spatio - Temporal Visual Information, 84-7, University of Toronto, Research in Biological and COmputational Vision, September 1984.

[8] S. Geman and D. Geman, Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images, *PAMI* 6.6 (Novermber 1984), 721-741, IEEE.

[9] X. Li and R. C. Dubes, The FIrst Stage in Two-Stage Template Matching, *Pattern Analysis and Machine Intelligence* 7.6 (Nov 1985), 700-707, IEEE.

[10] W. H. H. J. Lunscher and M. P. Beddoes, Optimal Edge Detector Design I: Parameter Selection and Noise Effects, *Pattern Analysis and Machine Intelligence* 8.2 (March 1986), 164-177, IEEE.

[11] V. S. Nalwa, On Detecting Edges, *Proceedings: Image Understanding Workshop*, October 1984, 157-164.

[12] A. Owen, A neighbourhood-based classifier for LANDSAT data, *The Canadian Journal of Statistics 12*.3 (September 1984), 191-200, Statistical Society of Canada.

[13] D. Sher, Developing and Analyzing Boundary Detection Operators Using Probabilistic Models, *Workshop on Probability and Uncertainty in Artificial Intelligence*, August 1985.

[14] D. B. Sher, Evidence Combination for Vision using Likelihood Generators, *Proceedings: Image Understanding Workshop (DARPA)*, December 1985, 255-270. Sponsored by: Information Processing Techniques Office Defence Advanced Research Projects Agency.

[15] V. Torre and T. A. Poggio. On Edge Detection. *Pattern Analysis and MAchine Intelligence* 8.2 (March 1986), 147-163. IEEE.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.

END
DATE
FILMED
DTIC
10-88